

groff The GNU implementation of troff
Edition 1.22.3
Autumn 2014

*by Trent A. Fisher
and Werner Lemberg (@email{bug-groff@gnu.org})*

This manual documents GNU troff version 1.22.3.

Copyright © 1994--2014 Free Software Foundation, Inc.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, with the Front-Cover texts being “A GNU Manual,” and with the Back-Cover Texts as in (a) below. A copy of the license is included in the section entitled “GNU Free Documentation License.”

(a) The FSF’s Back-Cover Text is: “You have the freedom to copy and modify this GNU manual. Buying copies from the FSF supports it in developing GNU and promoting software freedom.”

18 May 2018

Table of Contents

1. Introduction	2
1.1. What Is <code>groff</code> ?	2
1.2. History	2
1.3. <code>groff</code> Capabilities	4
1.4. Macro Packages	4
1.5. Preprocessors	5
1.6. Output Devices	5
1.7. Credits	5
2. Invoking <code>groff</code>	6
2.1. Options	6
2.2. Environment	10
2.3. Macro Directories	11
2.4. Font Directories	11
2.5. Paper Size	12
2.6. Invocation Examples	12
2.6.1. <code>grog</code>	13
3. Tutorial for Macro Users	14
3.1. Basics	14
3.2. Common Features	15
3.2.1. Paragraphs	16
3.2.2. Sections and Chapters	16
3.2.3. Headers and Footers	16
3.2.4. Page Layout	17
3.2.5. Displays	17
3.2.6. Footnotes and Annotations	17
3.2.7. Table of Contents	17
3.2.8. Indices	17
3.2.9. Paper Formats	18
3.2.10. Multiple Columns	18
3.2.11. Font and Size Changes	18
3.2.12. Predefined Strings	18
3.2.13. Preprocessor Support	18
3.2.14. Configuration and Customization	18
4. Macro Packages	19
4.1. <code>man</code>	19
4.1.1. Options	19
4.1.2. Usage	20
4.1.3. Macros to set fonts	22
4.1.4. Miscellaneous macros	23
4.1.5. Predefined strings	23
4.1.6. Preprocessors in <code>man</code> pages	24
4.1.7. Optional <code>man</code> extensions	24
4.1.7. Custom headers and footers	24
4.1.7. Ultrix-specific <code>man</code> macros	24
4.1.7. Simple example	26

4.2. <code>mdoc</code>	26
4.3. <code>ms</code>	26
4.3.1. Introduction to <code>ms</code>	26
4.3.2. General structure of an <code>ms</code> document	26
4.3.3. Document control registers	27
4.3.3. Margin Settings	27
4.3.3. Text Settings	28
4.3.3. Paragraph Settings	29
4.3.3. Footnote Settings	29
4.3.3. Miscellaneous Number Registers	30
4.3.4. Cover page macros	31
4.3.5. Body text	32
4.3.5.1. Paragraphs	32
4.3.5.2. Headings	33
4.3.5.3. Highlighting	34
4.3.5.4. Lists	35
4.3.5.5. Indentation values	38
4.3.5.6. Tab Stops	38
4.3.5.7. Displays and keeps	38
4.3.5.8. Tables, figures, equations, and references	40
4.3.5.9. An example multi-page table	40
4.3.5.10. Footnotes	41
4.3.6. Page layout	41
4.3.6.1. Headers and footers	41
4.3.6.2. Margins	42
4.3.6.3. Multiple columns	42
4.3.6.4. Creating a table of contents	43
4.3.6.5. Strings and Special Characters	44
4.3.7. Differences from AT&T <code>ms</code>	46
4.3.7.1. <code>troff</code> macros not appearing in <code>groff</code>	47
4.3.7.2. <code>groff</code> macros not appearing in AT&T <code>troff</code>	47
4.3.8. Naming Conventions	48
4.4. <code>me</code>	48
4.5. <code>mm</code>	48
4.6. <code>mom</code>	76
5. <code>groff</code> Reference	97
5.1. Text	97
5.1.1. Filling and Adjusting	97
5.1.2. Hyphenation	97
5.1.3. Sentences	97
5.1.4. Tab Stops	98
5.1.5. Implicit Line Breaks	98
5.1.6. Input Conventions	98
5.1.7. Input Encodings	99
5.2. Measurements	99
5.2.1. Default Units	100
5.3. Expressions	101
5.4. Identifiers	102
5.5. Embedded Commands	103

5.5.1. Requests	104
5.5.1.1. Request and Macro Arguments	105
5.5.2. Macros	106
5.5.3. Escapes	106
5.5.3.1. Comments	107
5.6. Registers	109
5.6.1. Setting Registers	109
5.6.2. Interpolating Registers	111
5.6.3. Auto-increment	111
5.6.4. Assigning Formats	112
5.6.5. Built-in Registers	113
5.7. Manipulating Filling and Adjusting	115
5.8. Manipulating Hyphenation	119
5.9. Manipulating Spacing	123
5.10. Tabs and Fields	126
5.10.1. Leaders	128
5.10.2. Fields	129
5.11. Character Translations	129
5.12. Troff and Nroff Mode	133
5.13. Line Layout	134
5.14. Line Control	136
5.15. Page Layout	137
5.16. Page Control	139
5.17. Fonts and Symbols	140
5.17.1. Changing Fonts	141
5.17.2. Font Families	142
5.17.3. Font Positions	144
5.17.4. Using Symbols	145
5.17.5. Character Classes	151
5.17.6. Special Fonts	152
5.17.7. Artificial Fonts	152
5.17.8. Ligatures and Kerning	154
5.18. Sizes	157
5.18.1. Changing Type Sizes	157
5.18.2. Fractional Type Sizes	159
5.19. Strings	160
5.20. Conditionals and Loops	164
5.20.1. Operators in Conditionals	164
5.20.2. if-else	166
5.20.3. while	167
5.21. Writing Macros	168
5.21.1. Copy-in Mode	171
5.21.2. Parameters	171
5.22. Page Motions	173
5.23. Drawing Requests	177
5.24. Traps	181
5.24.1. Page Location Traps	181
5.24.2. Diversion Traps	184
5.24.3. Input Line Traps	184

5.24.4. Blank Line Traps	184
5.24.5. Leading Spaces Traps	184
5.24.6. End-of-input Traps	185
5.25. Diversions	187
5.26. Environments	190
5.27. Suppressing output	192
5.28. Colors	193
5.29. I/O	194
5.30. Postprocessor Access	198
5.31. Miscellaneous	199
5.32. <code>gtroff</code> Internals	201
5.33. Debugging	203
5.33.1. Warnings	205
5.34. Implementation Differences	207
6. Preprocessors	210
6.1. <code>geqn</code>	210
6.2. <code>gtbl</code>	220
6.3. <code>gpic</code>	230
6.4. <code>ggrn</code>	239
6.5. <code>grap</code>	245
6.6. <code>gchem</code>	245
6.7. <code>grefer</code>	251
6.8. <code>gsoelim</code>	262
6.9. <code>preconv</code>	264
7. Output Devices	267
7.1. Special Characters	267
7.2. <code>grotty</code>	267
7.2.1. Invoking <code>grotty</code>	267
7.3. <code>grops</code>	267
7.3.1. Invoking <code>grops</code>	268
7.3.2. Embedding PostScript	268
7.4. <code>gropdf</code>	268
7.4.1. Invoking <code>gropdf</code>	269
7.4.2. Embedding PDF	269
7.5. <code>grodvi</code>	269
7.5.1. Invoking <code>grodvi</code>	270
7.6. <code>grolj4</code>	270
7.6.1. Invoking <code>grolj4</code>	270
7.7. <code>grolbp</code>	270
7.7.1. Invoking <code>grolbp</code>	271
7.8. <code>grohtml</code>	271
7.8.1. Invoking <code>grohtml</code>	271
7.8.2. <code>grohtml</code> specific registers and strings	272
7.9. <code>gxditview</code>	273
7.9.1. Invoking <code>gxditview</code>	273
8. File formats	274
8.1. <code>gtroff</code> Output	274
8.1.1. Language Concepts	274
8.1.1.1. Separation	274

8.1.1.2. Argument Units	275
8.1.1.3. Document Parts	275
8.1.2. Command Reference	276
8.1.2.1. Comment Command	276
8.1.2.2. Simple Commands	276
8.1.2.3. Graphics Commands	278
8.1.2.4. Device Control Commands	281
8.1.2.5. Obsolete Command	282
8.1.3. Intermediate Output Examples	283
8.1.4. Output Language Compatibility	285
8.2. Font Files	285
8.2.1. DESC File Format	285
8.2.2. Font File Format	288
9. Installation	291
10. Copying This Manual	292

1. Introduction

GNU `troff` (or `groff`) is a system for typesetting documents. `troff` is very flexible and has been used extensively for some thirty years. It is well entrenched in the UNIX community.

1.1. What Is `groff`?

`groff` belongs to an older generation of document preparation systems, which operate more like compilers than the more recent interactive WYSIWYG¹ systems. `groff` and its contemporary counterpart, `TEX`, both work using a *batch* paradigm: The input (or *source*) files are normal text files with embedded formatting commands. These files can then be processed by `groff` to produce a typeset document on a variety of devices.

`groff` should not be confused with a *word processor*, an integrated system of editor and text formatter. Also, many word processors follow the WYSIWYG paradigm discussed earlier.

Although WYSIWYG systems may be easier to use, they have a number of disadvantages compared to `troff`:

- They must be used on a graphics display to work on a document.
- Most of the WYSIWYG systems are either non-free or are not very portable.
- `troff` is firmly entrenched in all UNIX systems.
- It is difficult to have a wide range of capabilities within the confines of a GUI/window system.
- It is more difficult to make global changes to a document.

“GUIs normally make it simple to accomplish simple actions and impossible to accomplish complex actions.” --Doug Gwyn (22/Jun/91 in `comp.unix.wizards`)

1.2. History

`troff` can trace its origins back to a formatting program called `RUNOFF`, written by Jerry Saltzer, which ran on the CTSS (*Compatible Time Sharing System*, a project of MIT, the Massachusetts Institute of Technology) in the mid-sixties.² The name came from the use of the phrase “run off a document”, meaning to print it out. Bob Morris ported it to the 635 architecture and called the program `roff` (an abbreviation of `runoff`). It was rewritten as `rf` for the PDP-7 (before having UNIX), and at the same time (1969), Doug McIlroy rewrote an extended and simplified version of `roff` in the BCPL programming language.

In 1971, the UNIX developers wanted to get a PDP-11, and to justify the cost, proposed the development of a document formatting system for the AT&T patents division. This first formatting program was a reimplementations of McIlroy’s `roff`, written by J. F. Ossanna.

When they needed a more flexible language, a new version of `roff` called `nroff` (“Newer `roff`”) was written. It had a much more complicated syntax, but provided the basis for all future versions. When they got a Graphic Systems CAT Phototypesetter, Ossanna wrote a

¹ What You See Is What You Get

² Jerome H. Saltzer, a grad student then, later a Professor of Electrical Engineering, now retired. Saltzer’s PhD thesis was the first application for `RUNOFF` and is available from the MIT Libraries.

version of `nroff` that would drive it. It was dubbed `troff`, for “typesetter `roff`”, although many people have speculated that it actually means “Times `roff`” because of the use of the Times font family in `troff` by default. As such, the name `troff` is pronounced ‘t-roff’ rather than ‘trough’.

With `troff` came `nroff` (they were actually the same program except for some ‘`#ifdef`’s), which was for producing output for line printers and character terminals. It understood everything `troff` did, and ignored the commands that were not applicable (e.g. font changes).

Since there are several things that cannot be done easily in `troff`, work on several preprocessors began. These programs would transform certain parts of a document into `troff`, which made a very natural use of pipes in UNIX.

The `eqn` preprocessor allowed mathematical formulae to be specified in a much simpler and more intuitive manner. `tbl` is a preprocessor for formatting tables. The `refer` preprocessor (and the similar program, `bib`) processes citations in a document according to a bibliographic database.

Unfortunately, Ossanna’s `troff` was written in PDP-11 assembly language and produced output specifically for the CAT phototypesetter. He rewrote it in C, although it was now 7000 lines of uncommented code and still dependent on the CAT. As the CAT became less common, and was no longer supported by the manufacturer, the need to make it support other devices became a priority. However, before this could be done, Ossanna died by a severe heart attack in a hospital while recovering from a previous one.

So, Brian Kernighan took on the task of rewriting `troff`. The newly rewritten version produced device independent code that was very easy for postprocessors to read and translate to the appropriate printer codes. Also, this new version of `troff` (called `ditroff` for “device independent `troff`”) had several extensions, which included drawing functions.

Due to the additional abilities of the new version of `troff`, several new preprocessors appeared. The `pic` preprocessor provides a wide range of drawing functions. Likewise the `ideal` preprocessor did the same, although via a much different paradigm. The `grap` preprocessor took specifications for graphs, but, unlike other preprocessors, produced `pic` code.

James Clark began work on a GNU implementation of `ditroff` in early 1989. The first version, `groff` 0.3.1, was released June 1990. `groff` included:

- A replacement for `ditroff` with many extensions.
- The `soelim`, `pic`, `tbl`, and `eqn` preprocessors.
- Postprocessors for character devices, `POSTSCRIPT`, `TEX DVI`, and X Windows. GNU `troff` also eliminated the need for a separate `nroff` program with a postprocessor that would produce ASCII output.
- A version of the `me` macros and an implementation of the `man` macros.

Also, a front-end was included that could construct the, sometimes painfully long, pipelines required for all the post- and preprocessors.

Development of GNU `troff` progressed rapidly, and saw the additions of a replacement for `refer`, an implementation of the `ms` and `mm` macros, and a program to deduce how to format a document (`grog`).

It was declared a stable (i.e. non-beta) package with the release of version 1.04 around November 1991.

Beginning in 1999, `groff` has new maintainers (the package was an orphan for a few years). As a result, new features and programs like `grn`, a preprocessor for gremlin images, and an output device to produce HTML and XHTML have been added.

1.3. `groff` Capabilities

So what exactly is `groff` capable of doing? `groff` provides a wide range of low-level text formatting operations. Using these, it is possible to perform a wide range of formatting tasks, such as footnotes, table of contents, multiple columns, etc. Here's a list of the most important operations supported by `groff`:

- text filling, adjusting, and centering
- hyphenation
- page control
- font and glyph size control
- vertical spacing (e.g. double-spacing)
- line length and indenting
- macros, strings, diversions, and traps
- number registers
- tabs, leaders, and fields
- input and output conventions and character translation
- overstrike, bracket, line drawing, and zero-width functions
- local horizontal and vertical motions and the width function
- three-part titles
- output line numbering
- conditional acceptance of input
- environment switching
- insertions from the standard input
- input/output file switching
- output and error messages

1.4. Macro Packages

Since `groff` provides such low-level facilities, it can be quite difficult to use by itself. However, `groff` provides a *macro* facility to specify how certain routine operations (e.g. starting paragraphs, printing headers and footers, etc.) should be done. These macros can be

collected together into a *macro package*. There are a number of macro packages available; the most common (and the ones described in this manual) are `man`, `mdoc`, `me`, `ms`, and `mm`.

1.5. Preprocessors

Although `groff` provides most functions needed to format a document, some operations would be unwieldy (e.g. to draw pictures). Therefore, programs called *preprocessors* were written that understand their own language and produce the necessary `groff` operations. These preprocessors are able to differentiate their own input from the rest of the document via markers.

To use a preprocessor, UNIX pipes are used to feed the output from the preprocessor into `groff`. Any number of preprocessors may be used on a given document; in this case, the preprocessors are linked together into one pipeline. However, with `groff`, the user does not need to construct the pipe, but only tell `groff` what preprocessors to use.

`groff` currently has preprocessors for producing tables (`tbl`), typesetting equations (`eqn`), drawing pictures (`pic` and `grn`), processing bibliographies (`refer`), and drawing chemical structures (`chem`). An associated program that is useful when dealing with preprocessors is `soelim`.

A free implementation of `grap`, a preprocessor for drawing graphs, can be obtained as an extra package; `groff` can use `grap` also.

Unique to `groff` is the `preconv` preprocessor that enables `groff` to handle documents in various input encodings.

There are other preprocessors in existence, but, unfortunately, no free implementations are available. Among them is a preprocessor for drawing mathematical pictures (`ideal`).

1.6. Output Devices

`groff` actually produces device independent code that may be fed into a postprocessor to produce output for a particular device. Currently, `groff` has postprocessors for `POSTSCRIPT` devices, character terminals, X Windows (for previewing), `TEX` DVI format, HP LaserJet 4 and Canon LBP printers (which use `CAPSL`), `HTML`, `XHTML`, and `PDF`.

1.7. Credits

Large portions of this manual were taken from existing documents, most notably, the manual pages for the `groff` package by James Clark, and Eric Allman's papers on the `me` macro package.

The section on the `man` macro package is partly based on Susan G. Kleinmann's `groff_man` manual page written for the Debian GNU/Linux system.

Larry Kollar contributed the section in the `ms` macro package.

2. Invoking groff

This section focuses on how to invoke the `groff` front end. This front end takes care of the details of constructing the pipeline among the preprocessors, `gtroff` and the postprocessor.

It has become a tradition that GNU programs get the prefix ‘g’ to distinguish it from its original counterparts provided by the host (see [Environment](#), for more details). Thus, for example, `geqn` is GNU `eqn`. On operating systems like GNU/Linux or the Hurd, which don’t contain proprietary versions of `troff`, and on MS-DOS/MS-Windows, where `troff` and associated programs are not available at all, this prefix is omitted since GNU `troff` is the only used incarnation of `troff`. Exception: ‘groff’ is never replaced by ‘roff’.

In this document, we consequently say ‘gtroff’ when talking about the GNU `troff` program. All other implementations of `troff` are called A T&T `troff`, which is the common origin of all `troff` derivatives (with more or less compatible changes). Similarly, we say ‘gpic’, ‘geqn’, etc.

2.1. Options

`groff` normally runs the `gtroff` program and a postprocessor appropriate for the selected device. The default device is ‘ps’ (but it can be changed when `groff` is configured and built). It can optionally preprocess with any of `gpic`, `geqn`, `gtbl`, `ggrn`, `grap`, `gchem`, `grefer`, `gsoelim`, or `preconv`.

This section only documents options to the `groff` front end. Many of the arguments to `groff` are passed on to `gtroff`, therefore those are also included. Arguments to pre- or postprocessors can be found in [gpic](#), [geqn](#), [gtbl](#), [ggrn](#), [grefer](#), [gchem](#), [gsoelim](#), [preconv](#), [Invoking grotty](#), [Invoking grops](#), [Invoking gropdf](#), [Invoking grohtml](#), [Invoking grodvi](#), [Invoking grolj4](#), [Invoking grolbp](#), and [Invoking gxditview](#).

The command line format for `groff` is:

```
groff [ -abceghijklpstvzCEGNRSUVXZ ] [ -dcs ] [ -Darg ]
      [ -ffam ] [ -Fdir ] [ -Idir ] [ -Karg ]
      [ -Larg ] [ -mname ] [ -Mdir ] [ -num ]
      [ -olist ] [ -Parg ] [ -rcn ] [ -Tdev ]
      [ -wname ] [ -Wname ] [ files... ]
```

The command-line format for `gtroff` is as follows.

```
gtroff [ -abcivzCERU ] [ -dcs ] [ -ffam ] [ -Fdir ]
       [ -mname ] [ -Mdir ] [ -num ] [ -olist ]
       [ -rcn ] [ -Tname ] [ -wname ] [ -Wname ]
       [ files... ]
```

Obviously, many of the options to `groff` are actually passed on to `gtroff`.

Options without an argument can be grouped behind a single `-`. A filename `of-` denotes the standard input. It is possible to have whitespace between an option and its parameter.

The `grog` command can be used to guess the correct `groff` command to format a file.

Here’s the description of the command-line options:

‘-a’ Generate an ASCII approximation of the typeset output. The read-only register `.A` is then set to 1. See [Built-in Registers](#). A typical example is

```
groff -a -man -Tdvi troff.man | less
```

which shows how lines are broken for the DVI device. Note that this option is rather useless today since graphic output devices are available virtually everywhere.

- '-b' Print a backtrace with each warning or error message. This backtrace should help track down the cause of the error. The line numbers given in the backtrace may not always be correct: `gtroff` can get confused by `as` or `am` requests while counting line numbers.
- '-c' Suppress color output.
- '-C' Enable compatibility mode. See [Implementation Differences](#), for the list of incompatibilities between `groff` and AT&T `troff`.
- '-dcs'
- '-dname=s' Define *c* or *name* to be a string *s*. *c* must be a one-letter name; *name* can be of arbitrary length. All string assignments happen before loading any macro file (including the start-up file).
- '-Darg' Set default input encoding used by `preconv` to *arg*. Implies `-k`.
- '-e' Preprocess with `geqn`.
- '-E' Inhibit all error messages.
- '-ffam' Use *fam* as the default font family. See [Font Families](#).
- '-Fdir' Search *dir* for subdirectories *devname* (*name* is the name of the device), for the `DESC` file, and for font files before looking in the standard directories (see [Font Directories](#)). This option is passed to all pre- and postprocessors using the `GROFF_FONT_PATH` environment variable.
- '-g' Preprocess with `ggrn`.
- '-G' Preprocess with `grap`. Implies `-p`.
- '-h' Print a help message.
- '-i' Read the standard input after all the named input files have been processed.
- '-ldir' This option may be used to specify a directory to search for files. It is passed to the following programs:
 - `gsoelim` (see [gsoelim](#) for more details); it also implies `groff`'s `-s` option.
 - `gtroff`; it is used to search files named in the `psbb` and `so` requests.
 - `grops`; it is used to search files named in the `\X'ps: import` and `\X'ps: file escapes`.

The current directory is always searched first. This option may be specified more than once; the directories are searched in the order specified. No directory search is performed for files specified using an absolute path.

- j Preprocess with `gchem`. Implies `-p`.
- k Preprocess with `preconv`. This is run before any other preprocessor. Please refer to `preconv`'s manual page for its behaviour if no `-K` (or `-D`) option is specified.
- Karg Set input encoding used by `preconv` to *arg*. Implies `-k`.
- l Send the output to a spooler for printing. The command used for this is specified by the `print` command in the device description file (see [Font Files](#), for more info). If not present, `-l` is ignored.
- Larg Pass *arg* to the spooler. Each argument should be passed with a separate `-L` option. Note that `groff` does not prepend a '-' to *arg* before passing it to the postprocessor. If the `print` keyword in the device description file is missing, `-L` is ignored.
- mname Read in the file *name.tmac*. Normally `groff` searches for this in its macro directories. If it isn't found, it tries `tmac.name` (searching in the same directories).
- Mdir Search directory *dir* for macro files before the standard directories (see [Macro Directories](#)).
- nnum Number the first page *num*.
- N Don't allow newlines with `eqn` delimiters. This is the same as the `-N` option in `geqn`.
- olist Output only pages in *list*, which is a comma-separated list of page ranges; '*n*' means print page *n*, '*m-n*' means print every page between *m* and *n*, '*-n*' means print every page up to *n*, '*n-*' means print every page beginning with *n*. `gtroff` exits after printing the last page in the list. All the ranges are inclusive on both ends.

Within `gtroff`, this information can be extracted with the '.P' register. See [Built-in Registers](#).

If your document restarts page numbering at the beginning of each chapter, then `gtroff` prints the specified page range for each chapter.
- p Preprocess with `gpic`.
- Parg Pass *arg* to the postprocessor. Each argument should be passed with a separate `-P` option. Note that `groff` does not prepend '-' to *arg* before passing it to the postprocessor.
- rcn Set number register *c* or *name* to the value *n*. *c* must be a one-letter name; *name* can be of arbitrary length. *n* can be any `gtroff` numeric expression. All register assignments happen before loading any macro file (including the start-up file).
- rname=*n*

-R Preprocess with `grefer`. No mechanism is provided for passing arguments to `grefer` because most `grefer` options have equivalent commands that can be included in the file. See [grefer](#), for more details.

Note that `gtroff` also accepts a `-R` option, which is not accessible via `groff`. This option prevents the loading of the `troffrc` and `troffrc-end` files.

-s Preprocess with `gsoelim`.

-S Safer mode. Pass the `-S` option to `gpic` and disable the `open`, `opena`, `pso`, `sy`, and `pi` requests. For security reasons, this is enabled by default.

-t Preprocess with `gtbl`.

-Tdev Prepare output for device *dev*. The default device is 'ps', unless changed when `groff` was configured and built. The following are the output devices currently available:

`ps` For POSTSCRIPT printers and previewers.

`pdf` For PDF viewers or printers.

`dvi` For T_EX DVI format.

`x75` For a 75 dpi X11 previewer.

`x75-12` For a 75 dpi X11 previewer with a 12 pt base font in the document.

`x100` For a 100 dpi X11 previewer.

`x100-12` For a 100 dpi X11 previewer with a 12 pt base font in the document.

`ascii` For typewriter-like devices using the (7-bit) ASCII character set.

`latin1` For typewriter-like devices that support the Latin-1 (ISO 8859-1) character set.

`utf8` For typewriter-like devices that use the Unicode (ISO 10646) character set with UTF-8 encoding.

`cp1047` For typewriter-like devices that use the EBCDIC encoding IBM cp1047.

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

`lbp` For Canon CAPSL printers (LBP-4 and LBP-8 series laser printers).

`html`

`xhtml` To produce HTML and XHTML output, respectively. Note that this driver consists of two parts, a preprocessor (`pre-grohtml`) and a postprocessor (`post-grohtml`).

The predefined `gtroff` string register `.T` contains the current output device; the read-only number register `.T` is set to 1 if this option is used (which is always true if `groff` is used to call `gtroff`). See [Built-in Registers](#).

The postprocessor to be used for a device is specified by the `postpro`

command in the device description file. (See [Font Files](#), for more info.) This can be overridden with the `-X` option.

- `-U` Unsafe mode. This enables the `open`, `opena`, `pso`, `sy`, and `pi` requests.
- `-wname` Enable warning *name*. Available warnings are described in [Debugging](#). Multiple `-w` options are allowed.
- `-Wname` Inhibit warning *name*. Multiple `-W` options are allowed.
- `-v` Make programs run by `groff` print out their version number.
- `-V` Print the pipeline on `stdout` instead of executing it. If specified more than once, print the pipeline on `stderr` and execute it.
- `-X` Preview with `gxditview` instead of using the usual postprocessor. This is unlikely to produce good results except with `-Tps`.
Note that this is not the same as using `-TX75` or `-TX100` to view a document with `gxditview`: The former uses the metrics of the specified device, whereas the latter uses X-specific fonts and metrics.
- `-z` Suppress output from `gtroff`. Only error messages are printed.
- `-Z` Do not postprocess the output of `gtroff`. Normally `groff` automatically runs the appropriate postprocessor.

2.2. Environment

There are also several environment variables (of the operating system, not within `gtroff`) that can modify the behavior of `groff`.

GROFF_BIN_PATH

This search path, followed by `PATH`, is used for commands executed by `groff`.

GROFF_COMMAND_PREFIX

If this is set to `X`, then `groff` runs `Xtroff` instead of `gtroff`. This also applies to `tbl`, `pic`, `eqn`, `grn`, `chem`, `refer`, and `soelim`. It does not apply to `grops`, `grodvi`, `grotty`, `pre-grohtml`, `post-grohtml`, `preconv`, `grolj4`, `gropdf`, and `gxditview`.

The default command prefix is determined during the installation process. If a non-GNU troff system is found, prefix 'g' is used, none otherwise.

GROFF_ENCODING

The value of this environment value is passed to the `preconv` preprocessor to select the encoding of input files. Setting this option implies `groff`'s command-line option `-k` (that is, `groff` actually always calls `preconv`). If set without a value, `groff` calls `preconv` without arguments. An explicit `-K` command-line option overrides the value of `GROFF_ENCODING`. See the manual page of `preconv` for details.

GROFF_FONT_PATH

A colon-separated list of directories in which to search for the `devname` directory (before the default directories are tried). See [Font Directories](#).

GROFF_TMAC_PATH

A colon-separated list of directories in which to search for macro files (before the default directories are tried). See [Macro Directories](#).

GROFF_TMPDIR

The directory in which `groff` creates temporary files. If this is not set and `TMPPDIR` is set, temporary files are created in that directory. Otherwise temporary files are created in a system-dependent default directory (on Unix and GNU/Linux systems, this is usually `/tmp`). `grops`, `grefer`, `pre-grohtml`, and `post-grohtml` can create temporary files in this directory.

GROFF_TTYPESETTER

The default output device.

SOURCE_DATE_EPOCH

A timestamp (expressed as seconds since the Unix epoch) to use in place of the current time when initializing time-based built-in registers such as `\n[seconds]`.

Note that MS-DOS and MS-Windows ports of `groff` use semi-colons, rather than colons, to separate the directories in the lists described above.

2.3. Macro Directories

All macro file names must be named `name.tmac` or `tmac.name` to make the `-mname` command-line option work. The `ms0` request doesn't have this restriction; any file name can be used, and `gtroff` won't try to append or prepend the 'tmac' string.

Macro files are kept in the *tmac directories*, all of which constitute the *tmac path*. The elements of the search path for macro files are (in that order):

- The directories specified with `gtroff`'s or `groff`'s `-M` command-line option.
- The directories given in the `GROFF_TMAC_PATH` environment variable.
- The current directory (only if in unsafe mode using the `-U` command-line switch).
- The home directory.
- A platform-dependent directory, a site-specific (platform-independent) directory, and the main tmac directory; the default locations are

```
/usr/local/lib/groff/site-tmac
/usr/local/share/groff/site-tmac
/usr/local/share/groff/1.22.3/tmac
```

assuming that the version of `groff` is 1.22.3, and the installation prefix was `/usr/local`. It is possible to fine-tune those directories during the installation process.

2.4. Font Directories

Basically, there is no restriction how font files for `groff` are named and how long font names are; however, to make the font family mechanism work (see [Font Families](#)), fonts within a family should start with the family name, followed by the shape. For example, the

Times family uses ‘T’ for the family name and ‘R’, ‘B’, ‘I’, and ‘BI’ to indicate the shapes ‘roman’, ‘bold’, ‘italic’, and ‘bold italic’, respectively. Thus the final font names are ‘TR’, ‘TB’, ‘TI’, and ‘TBI’.

All font files are kept in the *font directories*, which constitute the *font path*. The file search functions always append the directory *devname*, where *name* is the name of the output device. Assuming, say, DVI output, and `/foo/bar` as a font directory, the font files for `grodvi` must be in `/foo/bar/devdvi`.

The elements of the search path for font files are (in that order):

- The directories specified with `gtroff`’s or `groff`’s `-F` command-line option. All device drivers and some preprocessors also have this option.
- The directories given in the `GROFF_FONT_PATH` environment variable.
- A site-specific directory and the main font directory; the default locations are

```
/usr/local/share/groff/site-font
/usr/local/share/groff/1.22.3/font
```

assuming that the version of `groff` is 1.22.3, and the installation prefix was `/usr/local`. It is possible to fine-tune those directories during the installation process.

2.5. Paper Size

In `groff`, the page size for `gtroff` and for output devices are handled separately. See [Page Layout](#), for vertical manipulation of the page size. See [Line Layout](#), for horizontal changes.

A default paper size can be set in the device’s `DESC` file. Most output devices also have a command-line option `-p` to override the default paper size and option `-l` to use landscape orientation. See [DESC File Format](#), for a description of the `papersize` keyword, which takes the same argument as `-p`.

A convenient shorthand to set a particular paper size for `gtroff` is command-line option `-dpaper=size`. This defines string `paper`, which is processed in file `papersize.tmac` (loaded in the start-up file `troffrc` by default). Possible values for *size* are the same as the predefined values for the `papersize` keyword (but only in lowercase) except `a7--d7`. An appended ‘l’ (ell) character denotes landscape orientation.

For example, use the following for PS output on A4 paper in landscape orientation:

```
groff -Tps -dpaper=a4l -P-pa4 -P-l -ms foo.ms > foo.ps
```

Note that it is up to the particular macro package to respect default page dimensions set in this way (most do).

2.6. Invocation Examples

This section lists several common uses of `groff` and the corresponding command lines.

```
groff file
```

This command processes *file* without a macro package or a preprocessor. The output device is the default, ‘ps’, and the output is sent to `stdout`.

```
groff -t -mandoc -Tascii file | less
```

This is basically what a call to the `man` program does. `gtroff` processes the manual page `file` with the `mandoc` macro file (which in turn either calls the `man` or the `mdoc` macro package), using the `tbl` preprocessor and the ASCII output device. Finally, the `less` pager displays the result.

```
groff -X -m me file
```

Preview `file` with `gxditview`, using the `me` macro package. Since no `-T` option is specified, use the default device ('ps'). Note that you can either say '-m me' or '-me'; the latter is an anachronism from the early days of UNIX.³

```
groff -man -rD1 -z file
```

Check `file` with the `man` macro package, forcing double-sided printing -- don't produce any output.

2.6.1. `grog`

`grog` reads files, guesses which of the `groff` preprocessors and/or macro packages are required for formatting them, and prints the `groff` command including those options on the standard output. It generates one or more of the options `-e`, `-man`, `-me`, `-mm`, `-mom`, `-ms`, `-mdoc`, `-mdoc-old`, `-p`, `-R`, `-g`, `-G`, `-s`, and `-t`.

A special file name `-` refers to the standard input. Specifying no files also means to read the standard input. Any specified options are included in the printed command. No space is allowed between options and their arguments. The only options recognized are `-C` (which is also passed on) to enable compatibility mode, and `-v` to print the version number and exit.

For example,

```
grog -Tdvi paper.ms
```

guesses the appropriate command to print `paper.ms` and then prints it to the command line after adding the `-Tdvi` option. For direct execution, enclose the call to `grog` in backquotes at the UNIX shell prompt:

```
`grog -Tdvi paper.ms` > paper.dvi
```

As seen in the example, it is still necessary to redirect the output to something meaningful (i.e. either a file or a pager program like `less`).

³ The same is true for the other main macro packages that come with `groff`: `man`, `mdoc`, `ms`, `mm`, and `mandoc`. This won't work in general; for example, to load `trace.tmac`, either '-mtrace' or '-m trace' must be used.

3. Tutorial for Macro Users

Most users tend to use a macro package to format their papers. This means that the whole breadth of `groff` is not necessary for most people. This chapter covers the material needed to efficiently use a macro package.

3.1. Basics

This section covers some of the basic concepts necessary to understand how to use a macro package.⁴ References are made throughout to more detailed information, if desired.

`gtroff` reads an input file prepared by the user and outputs a formatted document suitable for publication or framing. The input consists of text, or words to be printed, and embedded commands (*requests* and *escapes*), which tell `gtroff` how to format the output. For more detail on this, see [Embedded Commands](#).

The word *argument* is used in this chapter to mean a word or number that appears on the same line as a request, and which modifies the meaning of that request. For example, the request

```
.sp
```

spaces one line, but

```
.sp 4
```

spaces four lines. The number 4 is an argument to the `sp` request, which says to space four lines instead of one. Arguments are separated from the request and from each other by spaces (*no tabs*). More details on this can be found in [Request and Macro Arguments](#).

The primary function of `gtroff` is to collect words from input lines, fill output lines with those words, justify the right-hand margin by inserting extra spaces in the line, and output the result. For example, the input:

```
Now is the time
for all good men
to come to the aid
of their party.
Four score and seven
years ago, etc.
```

is read, packed onto output lines, and justified to produce:

```
Now is the time for all good men to come to the aid of their party. Four score and seven
years ago, etc.
```

Sometimes a new output line should be started even though the current line is not yet full; for example, at the end of a paragraph. To do this it is possible to cause a *break*, which starts a new output line. Some requests cause a break automatically, as normally do blank input lines and input lines beginning with a space.

Not all input lines are text to be formatted. Some input lines are requests that describe how to format the text. Requests always have a period (".") or an apostrophe (") as the first character of the input line.

The text formatter also does more complex things, such as automatically numbering pages, skipping over page boundaries, putting footnotes in the correct place, and so forth.

⁴ This section is derived from *Writing Papers with nroff using -me* by Eric P. Allman.

Here are a few hints for preparing text for input to `gtroff`.

- First, keep the input lines short. Short input lines are easier to edit, and `gtroff` packs words onto longer lines anyhow.
- In keeping with this, it is helpful to begin a new line after every comma or phrase, since common corrections are to add or delete sentences or phrases.
- End each sentence with two spaces -- or better, start each sentence on a new line. `gtroff` recognizes characters that usually end a sentence, and inserts sentence space accordingly.
- Do not hyphenate words at the end of lines -- `gtroff` is smart enough to hyphenate words as needed, but is not smart enough to take hyphens out and join a word back together. Also, words such as “mother-in-law” should not be broken over a line, since then a space can occur where not wanted, such as “mother-in-law”.

`gtroff` double-spaces output text automatically if you use the request `‘.ls 2’`. Reactivate single-spaced mode by typing `‘.ls 1’`.⁵

A number of requests allow to change the way the output looks, sometimes called the *layout* of the output page. Most of these requests adjust the placing of *whitespace* (blank lines or spaces).

The `bp` request starts a new page, causing a line break.

The request `‘.sp N’` leaves N lines of blank space. N can be omitted (meaning skip a single line) or can be of the form M_i (for N inches) or N_c (for N centimeters). For example, the input:

```
.sp 1.5i
My thoughts on the subject
.sp
```

leaves one and a half inches of space, followed by the line “My thoughts on the subject”, followed by a single blank line (more measurement units are available, see [Measurements](#)).

Text lines can be centered by using the `ce` request. The line after `ce` is centered (horizontally) on the page. To center more than one line, use `‘.ce N’` (where N is the number of lines to center), followed by the N lines. To center many lines without counting them, type:

```
.ce 1000
lines to center
.ce 0
```

The `‘.ce 0’` request tells `gtroff` to center zero more lines, in other words, stop centering.

All of these requests cause a break; that is, they always start a new line. To start a new line without performing any other action, use `br`.

3.2. Common Features

`gtroff` provides very low-level operations for formatting a document. There are many common routine operations that are done in all documents. These common operations are

⁵ If you need finer granularity of the vertical space, use the `pvs` request (see [Changing Type Sizes](#)).

written into *macros* and collected into a *macro package*.

All macro packages provide certain common capabilities that fall into the following categories.

3.2.1. Paragraphs

One of the most common and most used capability is starting a paragraph. There are a number of different types of paragraphs, any of which can be initiated with macros supplied by the macro package. Normally, paragraphs start with a blank line and the first line indented, like the text in this manual. There are also block style paragraphs, which omit the indentation:

```
Some men look at constitutions with sanctimonious
reverence, and deem them like the ark of the covenant, too
sacred to be touched.
```

And there are also indented paragraphs, which begin with a tag or label at the margin and the remaining text indented.

```
one This is the first paragraph. Notice how the first
line of the resulting paragraph lines up with the
other lines in the paragraph.
```

```
longlabel
```

```
This paragraph had a long label. The first
character of text on the first line does not line up
with the text on second and subsequent lines,
although they line up with each other.
```

A variation of this is a bulleted list.

```
. Bulleted lists start with a bullet. It is possible
to use other glyphs instead of the bullet. In nroff
mode using the ASCII character set for output, a dot
is used instead of a real bullet.
```

3.2.2. Sections and Chapters

Most macro packages supply some form of section headers. The simplest kind is simply the heading on a line by itself in bold type. Others supply automatically numbered section heading or different heading styles at different levels. Some, more sophisticated, macro packages supply macros for starting chapters and appendices.

3.2.3. Headers and Footers

Every macro package gives some way to manipulate the *headers* and *footers* (also called *titles*) on each page. This is text put at the top and bottom of each page, respectively, which contain data like the current page number, the current chapter title, and so on. Its appearance is not affected by the running text. Some packages allow for different ones on the even and odd pages (for material printed in a book form).

The titles are called *three-part titles*, that is, there is a left-justified part, a centered part, and a right-justified part. An automatically generated page number may be put in any of these fields with the '%' character (see [Page Layout](#), for more details).

3.2.4. Page Layout

Most macro packages let the user specify top and bottom margins and other details about the appearance of the printed pages.

3.2.5. Displays

Displays are sections of text to be set off from the body of the paper. Major quotes, tables, and figures are types of displays, as are all the examples used in this document.

Major quotes are quotes that are several lines long, and hence are set in from the rest of the text without quote marks around them.

A *list* is an indented, single-spaced, unfilled display. Lists should be used when the material to be printed should not be filled and justified like normal text, such as columns of figures or the examples used in this paper.

A *keep* is a display of lines that are kept on a single page if possible. An example for a keep might be a diagram. Keeps differ from lists in that lists may be broken over a page boundary whereas keeps are not.

Floating keeps move relative to the text. Hence, they are good for things that are referred to by name, such as “See figure 3”. A floating keep appears at the bottom of the current page if it fits; otherwise, it appears at the top of the next page. Meanwhile, the surrounding text ‘flows’ around the keep, thus leaving no blank areas.

3.2.6. Footnotes and Annotations

There are a number of requests to save text for later printing.

Footnotes are printed at the bottom of the current page.

Delayed text is very similar to a footnote except that it is printed when called for explicitly. This allows a list of references to appear (for example) at the end of each chapter, as is the convention in some disciplines.

Most macro packages that supply this functionality also supply a means of automatically numbering either type of annotation.

3.2.7. Table of Contents

Tables of contents are a type of delayed text having a tag (usually the page number) attached to each entry after a row of dots. The table accumulates throughout the paper until printed, usually after the paper has ended. Many macro packages provide the ability to have several tables of contents (e.g. a standard table of contents, a list of tables, etc).

3.2.8. Indices

While some macro packages use the term *index*, none actually provide that functionality. The facilities they call indices are actually more appropriate for tables of contents.

To produce a real index in a document, external tools like the `makeindex` program are necessary.

3.2.9. Paper Formats

Some macro packages provide stock formats for various kinds of documents. Many of them provide a common format for the title and opening pages of a technical paper. The `mm` macros in particular provide formats for letters and memoranda.

3.2.10. Multiple Columns

Some macro packages (but not `man`) provide the ability to have two or more columns on a page.

3.2.11. Font and Size Changes

The built-in font and size functions are not always intuitive, so all macro packages provide macros to make these operations simpler.

3.2.12. Predefined Strings

Most macro packages provide various predefined strings for a variety of uses; examples are sub- and superscripts, printable dates, quotes and various special characters.

3.2.13. Preprocessor Support

All macro packages provide support for various preprocessors and may extend their functionality.

For example, all macro packages mark tables (which are processed with `gtbl`) by placing them between `TS` and `TE` macros. The `ms` macro package has an option, `'.TS H'`, that prints a caption at the top of a new page (when the table is too long to fit on a single page).

3.2.14. Configuration and Customization

Some macro packages provide means of customizing many of the details of how the package behaves. This ranges from setting the default type size to changing the appearance of section headers.

4. Macro Packages

This chapter documents the main macro packages that come with `groff`.

Different main macro packages can't be used at the same time; for example

```
groff -m man foo.man -m ms bar.doc
```

doesn't work. Note that option arguments are processed before non-option arguments; the above (failing) sample is thus reordered to

```
groff -m man -m ms foo.man bar.doc
```

4.1. man

This is the most popular and probably the most important macro package of `groff`. It is easy to use, and a vast majority of manual pages are based on it.

4.1.1. Options

The command-line format for using the `man` macros with `groff` is:

```
groff -m man [ -rLL=length ] [ -rLT=length ] [ -rFT=dist ]
  [ -rcR=1 ] [ -rC1 ] [ -rD1 ] [ -rHY=flags ]
  [ -rPnnn ] [ -rSXX ] [ -rXnnn ]
  [ -rIN=length ] [ -rSN=length ] [ files... ]
```

It is possible to use '-man' instead of '-m man'.

`-rcR=1` This option (the default if a TTY output device is used) creates a single, very long page instead of multiple pages. Use `-rcR=0` to disable it.

`-rC1` If more than one manual page is given on the command line, number the pages continuously, rather than starting each at 1.

`-rD1` Double-sided printing. Footers for even and odd pages are formatted differently.

`-rFT=dist`

Set the position of the footer text to *dist*. If positive, the distance is measured relative to the top of the page, otherwise it is relative to the bottom. The default is `-0.5i`.

`-rHY=flags`

Set hyphenation flags. Possible values are 1 to hyphenate without restrictions, 2 to not hyphenate the last word on a page, 4 to not hyphenate the last two characters of a word, and 8 to not hyphenate the first two characters of a word. These values are additive; the default is 8.

`-rIN=length`

Set the body text indentation to *length*. If not specified, the indentation defaults to `7n` (7 characters) in `nroff` mode and `7.2n` otherwise. For `nroff`, this value should always be an integer multiple of unit 'n' to get consistent indentation.

`-rLL=length`

Set line length to *length*. If not specified, the line length is set to respect any value set by a prior 'll' request (which *must* be in effect when the 'TH' macro is invoked), if this differs from the built-in default for the formatter; otherwise it

defaults to 78n in nroff mode (this is 78 characters per line) and 6.5i in troff mode.⁶

`-rLT=length`

Set title length to *length*. If not specified, the title length defaults to the line length.

`-rPnnn` Page numbering starts with *nnn* rather than with 1.

`-rSxx` Use *xx* (which can be 10, 11, or 12 pt) as the base document font size instead of the default value of 10 pt.

`-rSN=length`

Set the indentation for sub-subheadings to *length*. If not specified, the indentation defaults to 3n.

`-rXnnn` After page *nnn*, number pages as *nnna*, *nnnb*, *nnnc*, etc. For example, the option `-rX2` produces the following page numbers: 1, 2, 2a, 2b, 2c, etc.

4.1.2. Usage

This section describes the available macros for manual pages. For further customization, put additional macros and requests into the file `man.local`, which is loaded immediately after the `man` package.

`.TH title section [extra1 [extra2 [extra3]]]`

Set the title of the man page to *title* and the section to *section*, which must have a value between 1 and 8. The value of *section* may also have a string appended, e.g. `‘.pm’`, to indicate a specific subsection of the man pages.

Both *title* and *section* are positioned at the left and right in the header line (with *section* in parentheses immediately appended to *title*). *extra1* is positioned in the middle of the footer line. *extra2* is positioned at the left in the footer line (or at the left on even pages and at the right on odd pages if double-sided printing is active). *extra3* is centered in the header line.

For HTML and XHTML output, headers and footers are completely suppressed.

Additionally, this macro starts a new page; the new line number is 1 again (except if the `-rC1` option is given on the command line) -- this feature is intended only for formatting multiple man pages; a single man page should contain exactly one `TH` macro at the beginning of the file.

`.SH [heading]`

Set up an unnumbered section heading sticking out to the left. Prints out all the text following `SH` up to the end of the line (or the text in the next line if there is no argument to `SH`) in bold face (or the font specified by the string `HF`), one size larger than the base document size. Additionally, the left margin and the indentation for the

⁶ Note that the use of a `‘.ll length’` request to initialize the line length, prior to use of the `‘TH’` macro, is supported for backward compatibility with some versions of the `man` program. Always use the `-rLL=length` option, or an equivalent `‘.nr LL length’` request, in preference to such a `‘.ll length’` request. In particular, note that in nroff mode, the request `‘.ll 65n’`, (with any *length* expression that evaluates equal to 65n, i.e., the formatter’s default line length in nroff mode), does *not* set the line length to 65n (it is adjusted to the `man` macro package’s default setting of 78n), whereas the use of the `-rLL=65n` option, or the `‘.nr LL 65n’` request *does* establish a line length of 65n.

following text is reset to its default value.

`.SS` [*heading*]

Set up an unnumbered (sub)section heading. Prints out all the text following `SS` up to the end of the line (or the text in the next line if there is no argument to `SS`) in bold face (or the font specified by the string `HF`), at the same size as the base document size. Additionally, the left margin and the indentation for the following text is reset to its default value.

`.TP` [*nnn*]

Set up an indented paragraph with label. The indentation is set to *nnn* if that argument is supplied (the default unit is ‘n’ if omitted), otherwise it is set to the previous indentation value specified with `TP`, `IP`, or `HP` (or to the default value if none of them have been used yet).

The first line of text following this macro is interpreted as a string to be printed flush-left, as it is appropriate for a label. It is not interpreted as part of a paragraph, so there is no attempt to fill the first line with text from the following input lines. Nevertheless, if the label is not as wide as the indentation the paragraph starts at the same line (but indented), continuing on the following lines. If the label is wider than the indentation the descriptive part of the paragraph begins on the line following the label, entirely indented. Note that neither font shape nor font size of the label is set to a default value; on the other hand, the rest of the text has default font settings.

`.LP`

`.PP`

`.P`

These macros are mutual aliases. Any of them causes a line break at the current position, followed by a vertical space downwards by the amount specified by the `PD` macro. The font size and shape are reset to the default value (10 pt roman if no `-rS` option is given on the command line). Finally, the current left margin and the indentation is restored.

`.IP` [*designator* [*nnn*]]

Set up an indented paragraph, using *designator* as a tag to mark its beginning. The indentation is set to *nnn* if that argument is supplied (default unit is ‘n’), otherwise it is set to the previous indentation value specified with `TP`, `IP`, or `HP` (or the default value if none of them have been used yet). Font size and face of the paragraph (but not the designator) are reset to their default values.

To start an indented paragraph with a particular indentation but without a designator, use “” (two double quotes) as the first argument of `IP`.

For example, to start a paragraph with bullets as the designator and 4 en indentation, write

```
.IP \ (bu 4
```

`.HP` [*nnn*]

Set up a paragraph with hanging left indentation. The indentation is set to *nnn* if that argument is supplied (default unit is ‘n’), otherwise it is set to the previous indentation value specified with `TP`, `IP`, or `HP` (or the default value if non of them have been used yet). Font size and face are reset to their default values.

`.RS` [*nnn*]

Move the left margin to the right by the value *nnn* if specified (default unit is ‘n’);

otherwise it is set to the previous indentation value specified with TP, IP, or HP (or to the default value if none of them have been used yet). The indentation value is then set to the default.

Calls to the RS macro can be nested.

`.RE [nnn]`

Move the left margin back to level *nnn*, restoring the previous left margin. If no argument is given, it moves one level back. The first level (i.e., no call to RS yet) has number 1, and each call to RS increases the level by 1.

To summarize, the following macros cause a line break with the insertion of vertical space (which amount can be changed with the PD macro): SH, SS, TP, LP (PP, P), IP, and HP.

The macros RS and RE also cause a break but do not insert vertical space.

Finally, the macros SH, SS, LP (PP, P), and RS reset the indentation to its default value.

4.1.3. Macros to set fonts

The standard font is roman; the default text size is 10 points. If command-line option `-rS=n` is given, use *n* points as the default text size.

`.SM [text]`

Set the text on the same line or the text on the next line in a font that is one point size smaller than the default font.

`.SB [text]`

Set the text on the same line or the text on the next line in bold face font, one point size smaller than the default font.

`.BI text`

Set its arguments alternately in bold face and italic, without a space between the arguments. Thus,

```
.BI this "word and" that
```

produces “thisword andthat” with “this” and “that” in bold face, and “word and” in italics.

`.IB text`

Set its arguments alternately in italic and bold face, without a space between the arguments.

`.RI text`

Set its arguments alternately in roman and italic, without a space between the arguments.

`.IR text`

Set its arguments alternately in italic and roman, without a space between the arguments.

`.BR text`

Set its arguments alternately in bold face and roman, without a space between the arguments.

`.RB text`

Set its arguments alternately in roman and bold face, without a space between the arguments.

.B [*text*]

Set *text* in bold face. If no text is present on the line where the macro is called, then the text of the next line appears in bold face.

.I [*text*]

Set *text* in italic. If no text is present on the line where the macro is called, then the text of the next line appears in italic.

4.1.4. Miscellaneous macros

The default indentation is 7.2n in troff mode and 7n in nroff mode except for `grohtml`, which ignores indentation.

.DT

Set tabs every 0.5 inches. Since this macro is always executed during a call to the `TH` macro, it makes sense to call it only if the tab positions have been changed.

.PD [*nnn*]

Adjust the empty space before a new paragraph (or section). The optional argument gives the amount of space (default unit is 'v'); without parameter, the value is reset to its default value (1 line in nroff mode, 0.4v otherwise).

This affects the macros `SH`, `SS`, `TP`, `LP` (as well as `PP` and `P`), `IP`, and `HP`.

The following two macros are included for BSD compatibility.

.AT [*system* [*release*]]

Alter the footer for use with AT&T manpages. This command exists only for compatibility; don't use it. The first argument *system* can be:

3 7th Edition (the default)

4 System III

5 System V

An optional second argument *release* to `AT` specifies the release number (such as "System V Release 3").

.UC [*version*]

Alters the footer for use with BSD manpages. This command exists only for compatibility; don't use it. The argument can be:

3 3rd Berkeley Distribution (the default)

4 4th Berkeley Distribution

5 4.2 Berkeley Distribution

6 4.3 Berkeley Distribution

7 4.4 Berkeley Distribution

4.1.5. Predefined strings

The following strings are defined:

`*[S]`

Switch back to the default font size.

`*[HF]`

The typeface used for headings. The default is 'B'.

`*[R]`

The 'registered' sign.

`*[Tm]`

The 'trademark' sign.

`*[lq]`

`*[rq]`

Left and right quote. This is equal to `\(lq` and `\(rq`, respectively.

4.1.6. Preprocessors in `man` pages

If a preprocessor like `gtbl` or `geqn` is needed, it has become common usage to make the first line of the `man` page look like this:

```
' \" word
```

Note the single space character after the double quote. `word` consists of letters for the needed preprocessors: 'e' for `geqn`, 'r' for `grefer`, 't' for `gtbl`. Modern implementations of the `man` program read this first line and automatically call the right preprocessor(s).

4.1.7. Optional `man` extensions

Use the file `man.local` for local extensions to the `man` macros or for style changes.

Custom headers and footers

In `groff` versions 1.18.2 and later, you can specify custom headers and footers by redefining the following macros in `man.local`.

`.PT`

Control the content of the headers. Normally, the header prints the command name and section number on either side, and the optional fifth argument to `TH` in the center.

`.BT`

Control the content of the footers. Normally, the footer prints the page number and the third and fourth arguments to `TH`.

Use the `FT` number register to specify the footer position. The default is `-0.5i`.

Ultrix-specific `man` macros

The `groff` source distribution includes a file named `man.ultrix`, containing macros compatible with the Ultrix variant of `man`. Copy this file into `man.local` (or use the `mso` request to load it) to enable the following macros.

`.CT key`

Print '<CTRL/key>'.

`.CW`

Print subsequent text using the constant width (Courier) typeface.

`.VE`

End printing the change bar begun by `VS`.

Simple example

The following example `man.local` file alters the `SH` macro to add some extra vertical space before printing the heading. Headings are printed in Helvetica Bold.

```
.\" Make the heading fonts Helvetica
.ds HF HB
.
.\" Put more whitespace in front of headings.
.rn SH SH-orig
.de SH
.  if t .sp (u;\n[PD]*2)
.  SH-orig \$*
..
```

4.2. `mdoc`

See the `groff_mdoc(7)` man page (type `man groff_mdoc` at the command line).

4.3. `ms`

The `-ms` macros are suitable for reports, letters, books, user manuals, and so forth. The package provides macros for cover pages, section headings, paragraphs, lists, footnotes, pagination, and a table of contents.

4.3.1. Introduction to `ms`

The original `-ms` macros were included with AT&T `troff` as well as the `man` macros. While the `man` package is intended for brief documents that can be read on-line as well as printed, the `ms` macros are suitable for longer documents that are meant to be printed rather than read on-line.

The `ms` macro package included with `groff` is a complete, bottom-up re-implementation. Several macros (specific to AT&T or Berkeley) are not included, while several new commands are. See [Differences from AT&T `ms`](#), for more information.

4.3.2. General structure of an `ms` document

The `ms` macro package expects a certain amount of structure, but not as much as packages such as `man` or `mdoc`.

The simplest documents can begin with a paragraph macro (such as `LP` or `PP`), and consist of text separated by paragraph macros or even blank lines. Longer documents have a structure as follows:

Document type

If you invoke the `RP` (report) macro on the first line of the document, `groff` prints the cover page information on its own page; otherwise it prints the information on the first page with your document text immediately following. Other document formats found in AT&T `troff` are specific to AT&T or Berkeley, and

are not supported in `groff`.

Format and layout

By setting number registers, you can change your document's type (font and size), margins, spacing, headers and footers, and footnotes. See [Document control registers](#), for more details.

Cover page

A cover page consists of a title, the author's name and institution, an abstract, and the date.⁷ See [Cover page macros](#), for more details.

Body

Following the cover page is your document. You can use the `ms` macros to write reports, letters, books, and so forth. The package is designed for structured documents, consisting of paragraphs interspersed with headings and augmented by lists, footnotes, tables, and other common constructs. See [Body text](#), for more details.

Table of contents

Longer documents usually include a table of contents, which you can invoke by placing the `TC` macro at the end of your document. The `ms` macros have minimal indexing facilities, consisting of the `IX` macro, which prints an entry on standard error. Printing the table of contents at the end is necessary since `groff` is a single-pass text formatter, thus it cannot determine the page number of each section until that section has actually been set and printed. Since `ms` output is intended for hardcopy, you can manually relocate the pages containing the table of contents between the cover page and the body text after printing.

4.3.3. Document control registers

The following is a list of document control number registers. For the sake of consistency, set registers related to margins at the beginning of your document, or just after the `RP` macro. You can set other registers later in your document, but you should keep them together at the beginning to make them easy to find and edit as necessary.

Margin Settings

`\n[PO]`

Defines the page offset (i.e., the left margin). There is no explicit right margin setting; the combination of the `PO` and `LL` registers implicitly define the right margin width.

Effective: next page.

Default value: 1 i.

`\n[LL]`

Defines the line length (i.e., the width of the body text).

Effective: next paragraph.

Default: 6 i.

`\n[LT]`

Defines the title length (i.e., the header and footer width). This is usually the same as `LL`, but not necessarily.

⁷ Actually, only the title is required.

Effective: next paragraph.

Default: 6 i.

`\n[HM]`

Defines the header margin height at the top of the page.

Effective: next page.

Default: 1 i.

`\n[FM]`

Defines the footer margin height at the bottom of the page.

Effective: next page.

Default: 1 i.

Text Settings

`\n[PS]`

Defines the point size of the body text. If the value is larger than or equal to 1000, divide it by 1000 to get a fractional point size. For example, `'\nr PS 10250'` sets the document's point size to 10.25 p.

Effective: next paragraph.

Default: 10 p.

`\n[VS]`

Defines the space between lines (line height plus leading). If the value is larger than or equal to 1000, divide it by 1000 to get a fractional point size. Due to backwards compatibility, `VS` must be smaller than 40000 (this is 40.0 p).

Effective: next paragraph.

Default: 12 p.

`\n[PSINCR]`

Defines an increment in point size, which is applied to section headings at nesting levels below the value specified in `GROWPS`. The value of `PSINCR` should be specified in points, with the `p` scaling factor, and may include a fractional component; for example, `'\nr PSINCR 1.5p'` sets a point size increment of 1.5 p.

Effective: next section heading.

Default: 1 p.

`\n[GROWPS]`

Defines the heading level below which the point size increment set by `PSINCR` becomes effective. Section headings at and above the level specified by `GROWPS` are printed at the point size set by `PS`; for each level below the value of `GROWPS`, the point size is increased in steps equal to the value of `PSINCR`. Setting `GROWPS` to an `y` value less than 2 disables the incremental heading size feature.

Effective: next section heading.

Default: 0.

`\n[HY]`

Defines the hyphenation level. `HY` sets safely the value of the low-level `hy` register. Setting the value of `HY` to 0 is equivalent to using the `nh` request.

Effective: next paragraph.

Default: 6.

`\n[FAM]`

Defines the font family used to typeset the document.

Effective: next paragraph.

Default: as defined in the output device.

Paragraph Settings

`\n[PI]`

Defines the initial indentation of a (`PP` macro) paragraph.

Effective: next paragraph.

Default: 5 n.

`\n[PD]`

Defines the space between paragraphs.

Effective: next paragraph.

Default: 0.3 v.

`\n[QI]`

Defines the indentation on both sides of a quoted (`QP`, `QS`, and `QE` macros) paragraph.

Effective: next paragraph.

Default: 5 n.

`\n[PORPHANS]`

Defines the minimum number of initial lines of any paragraph that should be kept together, to avoid orphan lines at the bottom of a page. If a new paragraph is started close to the bottom of a page, and there is insufficient space to accommodate `PORPHANS` lines before an automatic page break, then the page break is forced, before the start of the paragraph.

Effective: next paragraph.

Default: 1.

`\n[HORPHANS]`

Defines the minimum number of lines of the following paragraph that should be kept together with any section heading introduced by the `NH` or `SH` macros. If a section heading is placed close to the bottom of a page, and there is insufficient space to accommodate both the heading and at least `HORPHANS` lines of the following paragraph, before an automatic page break, then the page break is forced before the heading.

Effective: next paragraph.

Default: 1.

Footnote Settings

`\n[FL]`

Defines the length of a footnote.

Effective: next footnote.

Default: $\backslash n[LL] * 5 / 6$.

$\backslash n[FI]$

Defines the footnote indentation.

Effective: next footnote.

Default: $2n$.

$\backslash n[FF]$

The footnote format:

- | | |
|---|--|
| 0 | Print the footnote number as a superscript; indent the footnote (default). |
| 1 | Print the number followed by a period (like 1.) and indent the footnote. |
| 2 | Like 1, without an indentation. |
| 3 | Like 1, but print the footnote number as a hanging paragraph. |

Effective: next footnote.

Default: 0.

$\backslash n[FPS]$

Defines the footnote point size. If the value is larger than or equal to 1000, divide it by 1000 to get a fractional point size.

Effective: next footnote.

Default: $\backslash n[PS] - 2$.

$\backslash n[FVS]$

Defines the footnote vertical spacing. If the value is larger than or equal to 1000, divide it by 1000 to get a fractional point size.

Effective: next footnote.

Default: $\backslash n[FPS] + 2$.

$\backslash n[FPD]$

Defines the footnote paragraph spacing.

Effective: next footnote.

Default: $\backslash n[PD] / 2$.

Miscellaneous Number Registers

$\backslash n[MINGW]$

Defines the minimum width between columns in a multi-column document.

Effective: next page.

Default: $2n$.

$\backslash n[DD]$

Sets the vertical spacing before and after a display, a `tbl` table, an `eqn` equation, or a `pic` image.

Effective: next paragraph.

Default: 0.5 v.

4.3.4. Cover page macros

Use the following macros to create a cover page for your document in the order shown.

`.RP [no]`

Specifies the report format for your document. The report format creates a separate cover page. The default action (no `RP` macro) is to print a subset of the cover page on page 1 of your document.

If you use the word `no` as an optional argument, `groff` prints a title page but does not repeat any of the title page information (title, author, abstract, etc.) on page 1 of the document.

`.P1`

(P-one) Prints the header on page 1. The default is to suppress the header.

`.DA [...]`

(optional) Prints the current date, or the arguments to the macro if any, on the title page (if specified) and in the footers. This is the default for `nroff`.

`.ND [...]`

(optional) Prints the current date, or the arguments to the macro if any, on the title page (if specified) but not in the footers. This is the default for `troff`.

`.TL`

Specifies the document title. `groff` collects text following the `TL` macro into the title, until reaching the author name or abstract.

`.AU`

Specifies the author's name, which appears on the line (or lines) immediately following. You can specify multiple authors as follows:

```
.AU
John Doe
.AI
University of West Bumblefuzz
.AU
Martha Buck
.AI
Monolithic Corporation
```

...

`.AI`

Specifies the author's institution. You can specify multiple institutions in the same way that you specify multiple authors.

`.AB [no]`

Begins the abstract. The default is to print the word `ABSTRACT`, centered and in italics, above the text of the abstract. The word `no` as an optional argument suppresses this heading.

`.AE`

Ends the abstract.

The following is example mark-up for a title page.

```
.RP
.TL
The Inevitability of Code Bloat
in Commercial and Free Software
.AU
J. Random Luser
.AI
University of West Bumblefuzz
.AB
This report examines the long-term growth
of the code bases in two large, popular software
packages; the free Emacs and the commercial
Microsoft Word.
While differences appear in the type or order
of features added, due to the different
methodologies used, the results are the same
in the end.
.PP
The free software approach is shown to be
superior in that while free software can
become as bloated as commercial offerings,
free software tends to have fewer serious
bugs and the added features are in line with
user demand.
.AE

... the rest of the paper follows ...
```

4.3.5. Body text

This section describes macros used to mark up the body of your document. Examples include paragraphs, sections, and other groups.

4.3.5.1. Paragraphs

The following paragraph types are available.

- .PP
Sets a paragraph with an initial indentation.
- .LP
Sets a paragraph without an initial indentation.
- .QP
Sets a paragraph that is indented at both left and right margins by the amount of the register `QI`. The effect is identical to the HTML `<BLOCKQUOTE>` element. The next paragraph or heading returns margins to normal. `QP` inserts vertical space of amount set by register `PD` before the paragraph.

`.QS`

`.QE`

These macros begin and end a quoted section. The `QI` register controls the amount of indentation. Both `QS` and `QE` insert inter-paragraph vertical space set by register `PD`. The text between `QS` and `QE` can be structured further by use of the macros `LP` or `PP`.

`.XP`

Sets a paragraph whose lines are indented, except for the first line. This is a Berkeley extension.

The following markup uses all four paragraph macros.

```
.NH 2
Cases used in the study
.LP
The following software and versions were
considered for this report.
.PP
For commercial software, we chose
.B "Microsoft Word for Windows" ,
starting with version 1.0 through the
current version (Word 2000).
.PP
For free software, we chose
.B Emacs ,
from its first appearance as a standalone
editor through the current version (v20).
See [Bloggs 2002] for details.
.QP
Franklin's Law applied to software:
software expands to outgrow both
RAM and disk space over time.
.LP
Bibliography:
.XP
Bloggs, Joseph R.,
.I "Everyone's a Critic" ,
Underground Press, March 2002.
A definitive work that answers all questions
and criticisms about the quality and usability of
free software.
```

The `PORPHANS` register (see [Document control registers](#)) operates in conjunction with each of these macros, to inhibit the printing of orphan lines at the bottom of any page.

4.3.5.2. Headings

Use headings to create a hierarchical structure for your document. The `ms` macros print headings in **bold**, using the same font family and point size as the body text.

The following describes the heading macros:

`.NH curr-level`

`.NH sle vel0 ...`

Numbered heading. The argument is either a numeric argument to indicate the level of the heading, or the letter *s* followed by numeric arguments to set the heading level explicitly.

If you specify heading levels out of sequence, such as invoking `.NH 3` after `.NH 1`, `groff` prints a warning on standard error.

`*[SN]`

`*[SN-DOT]`

`*[SN-NO-DOT]`

After invocation of `NH`, the assigned section number is made available in the strings `SN-DOT` (as it appears in a printed section heading with default formatting, followed by a terminating period), and `SN-NO-DOT` (with the terminating period omitted). The string `SN` is also defined, as an alias for `SN-DOT`; if preferred, you may redefine it as an alias for `SN-NO-DOT`, by including the initialization

```
.als SN SN-NO-DOT
```

at any time **before** you would like the change to take effect.

`*[SN-STYLE]`

You may control the style used to print section numbers, within numbered section headings, by defining an appropriate alias for the string `SN-STYLE`. The default style, in which the printed section number is followed by a terminating period, is obtained by defining the alias

```
.als SN-STYLE SN-DOT
```

If you prefer to omit the terminating period, from section numbers appearing in numbered section headings, you may define the alias

```
.als SN-STYLE SN-NO-DOT
```

Any such change in section numbering style becomes effective from the next use of `.NH`, following redefinition of the alias for `SN-STYLE`.

`.SH [match-level]`

Unnumbered subheading.

The optional *match-level* argument is a GNU extension. It is a number indicating the level of the heading, in a manner analogous to the *curr-level* argument to `.NH`. Its purpose is to match the point size, at which the heading is printed, to the size of a numbered heading at the same level, when the `GROWPS` and `PSINCR` heading size adjustment mechanism is in effect. See [Document control registers](#).

The `HORPHANS` register (see [Document control registers](#)) operates in conjunction with the `NH` and `SH` macros, to inhibit the printing of orphaned section headings at the bottom of any page.

4.3.5.3. Highlighting

The `ms` macros provide a variety of methods to highlight or emphasize text:

`.B [txt [post [pre]]]`

Sets its first argument in **bold type**. If you specify a second argument, `groff` prints

it in the previous font after the bold text, with no intervening space (this allows you to set punctuation after the highlighted text without highlighting the punctuation). Similarly, it prints the third argument (if any) in the previous font **before** the first argument. For example,

```
.B foo ) (
```

prints (**foo**).

If you give this macro no arguments, `groff` prints all text following in bold until the next highlighting, paragraph, or heading macro.

`.R [txt [post [pre]]]`

Sets its first argument in roman (or regular) type. It operates similarly to the `B` macro otherwise.

`.I [txt [post [pre]]]`

Sets its first argument in *italic type*. It operates similarly to the `B` macro otherwise.

`.CW [txt [post [pre]]]`

Sets its first argument in a `constant width face`. It operates similarly to the `B` macro otherwise.

`.BI [txt [post [pre]]]`

Sets its first argument in bold italic type. It operates similarly to the `B` macro otherwise.

`.BX [txt]`

Prints its argument and draws a box around it. If you want to box a string that contains spaces, use a digit-width space (`\0`).

`.UL [txt [post]]`

Prints its first argument with an underline. If you specify a second argument, `groff` prints it in the previous font after the underlined text, with no intervening space.

`.LG`

Prints all text following in larger type (two points larger than the current point size) until the next font size, highlighting, paragraph, or heading macro. You can specify this macro multiple times to enlarge the point size as needed.

`.SM`

Prints all text following in smaller type (two points smaller than the current point size) until the next type size, highlighting, paragraph, or heading macro. You can specify this macro multiple times to reduce the point size as needed.

`.NL`

Prints all text following in the normal point size (that is, the value of the `PS` register).

`* [{]`

`* [}]`

Text enclosed with `* {` and `* }` is printed as a superscript.

4.3.5.4. Lists

The `IP` macro handles duties for all lists.

`.IP [marker [width]]`

The *marker* is usually a bullet glyph (`\[bu]`) for unordered lists, a number (or auto-incrementing number register) for numbered lists, or a word or phrase for indented

(glossary-style) lists.

The *width* specifies the indentation for the body of each list item; its default unit is 'n'. Once specified, the indentation remains the same for all list items in the document until specified again.

The `PORPHANS` register (see [Document control registers](#)) operates in conjunction with the `IP` macro, to inhibit the printing of orphaned list markers at the bottom of any page.

The following is an example of a bulleted list.

```
A bulleted list:
.IP \[bu] 2
lawyers
.IP \[bu]
guns
.IP \[bu]
money
```

Produces:

```
A bulleted list:

o lawyers

o guns

o money
```

The following is an example of a numbered list.

```
.nr step 1 1
A numbered list:
.IP \n[step] 3
lawyers
.IP \n+[step]
guns
.IP \n+[step]
money
```

Produces:

```
A numbered list:

1. lawyers

2. guns

3. money
```

Note the use of the auto-incrementing number register in this example.

The following is an example of a glossary-style list.

```
A glossary-style list:
.IP lawyers 0.4i
Two or more attorneys.
```

```
.IP guns
Firearms, preferably
large-caliber.
.IP money
Gotta pay for those
lawyers and guns!
```

Produces:

A glossary-style list:

```
lawyers
    Two or more attorneys.

guns Firearms, preferably large-caliber.

money
    Gotta pay for those lawyers and guns!
```

In the last example, the `IP` macro places the definition on the same line as the term if it has enough space; otherwise, it breaks to the next line and starts the definition below the term. This may or may not be the effect you want, especially if some of the definitions break and some do not. The following examples show two possible ways to force a break.

The first workaround uses the `br` request to force a break after printing the term or label.

```
A glossary-style list:
.IP lawyers 0.4i
Two or more attorneys.
.IP guns
.br
Firearms, preferably large-caliber.
.IP money
Gotta pay for those lawyers and guns!
```

The second workaround uses the `\p` escape to force the break. Note the space following the escape; this is important. If you omit the space, `groff` prints the first word on the same line as the term or label (if it fits) **then** breaks the line.

```
A glossary-style list:
.IP lawyers 0.4i
Two or more attorneys.
.IP guns
\p Firearms, preferably large-caliber.
.IP money
Gotta pay for those lawyers and guns!
```

To set nested lists, use the `RS` and `RE` macros. See [Indentation values](#), for more information.

For example:

```
.IP \[bu] 2
Lawyers:
.RS
.IP \[bu]
Dewey,
.IP \[bu]
Cheatham,
.IP \[bu]
and Howe.
.RE
.IP \[bu]
Guns
```

Produces:

- o Lawyers:
 - o Dewey,
 - o Cheatham,
 - o and Howe.
- o Guns

4.3.5.5. Indentation values

In many situations, you may need to indent a section of text while still wrapping and filling. See [Lists](#), for an example of nested lists.

```
.RS
.RE
```

These macros begin and end an indented section. The `PI` register controls the amount of indentation, allowing the indented text to line up under hanging and indented paragraphs.

See [Displays and keeps](#), for macros to indentation and turn off filling.

4.3.5.6. Tab Stops

Use the `ta` request to define tab stops as needed. See [Tabs and Fields](#).

```
.TA
```

Use this macro to reset the tab stops to the default for `ms` (every 5n). You can redefine the `TA` macro to create a different set of default tab stops.

4.3.5.7. Displays and keeps

Use displays to show text-based examples or figures (such as code listings).

Displays turn off filling, so lines of code are displayed as-is without inserting `br` requests in between each line. Displays can be *kept* on a single page, or allowed to break across

pages.

.DS L
.LD
.DE

Left-justified display. The '.DS L' call generates a page break, if necessary, to keep the entire display on one page. TheLD macro allows the display to break across pages. TheDE macro ends the display.

.DS I
.ID
.DE

Indents the display as defined by the DI register. The '.DS I' call generates a page break, if necessary, to keep the entire display on one page. TheID macro allows the display to break across pages. TheDE macro ends the display.

.DS B
.BD
.DE

Sets a block-centered display: the entire display is left-justified, but indented so that the longest line in the display is centered on the page. The '.DS B' call generates a page break, if necessary, to keep the entire display on one page. TheBD macro allows the display to break across pages. TheDE macro ends the display.

.DS C
.CD
.DE

Sets a centered display: each line in the display is centered. The '.DS C' call generates a page break, if necessary, to keep the entire display on one page. TheCD macro allows the display to break across pages. TheDE macro ends the display.

.DS R
.RD
.DE

Right-justifies each line in the display. The '.DS R' call generates a page break, if necessary, to keep the entire display on one page. TheRD macro allows the display to break across pages. TheDE macro ends the display.

.Ds
.De

These two macros were formerly provided as aliases for DS and DE, respectively. They have been removed, and should no longer be used. The original implementations of DS and DE are retained, and should be used instead. X11 documents that actually use Ds and De always load a specific macro file from the X11 distribution (macros.t) that provides proper definitions for the two macros.

On occasion, you may want to *keep* other text together on a page. For example, you may want to keep two paragraphs together, or a paragraph that refers to a table (or list, or other item) immediately following. Thems macros provide the KS and KE macros for this purpose.

.KS
.KE

The KS macro begins a block of text to be kept on a single page, and the KE macro

ends the block.

.KF

.KE

Specifies a *floating keep*; if the keep cannot fit on the current page, `groff` holds the contents of the keep and allows text following the keep (in the source file) to fill in the remainder of the current page. When the page breaks, whether by an explicit `bp` request or by reaching the end of the page, `groff` prints the floating keep at the top of the new page. This is useful for printing large graphics or tables that do not need to appear exactly where specified.

You can also use the `ne` request to force a page break if there is not enough vertical space remaining on the page.

Use the following macros to draw a box around a section of text (such as a display).

.B1

.B2

Marks the beginning and ending of text that is to have a box drawn around it. The `B1` macro begins the box; the `B2` macro ends it. Text in the box is automatically placed in a diversion (*keep*).

4.3.5.8. Tables, figures, equations, and references

The `ms` macros support the standard `groff` preprocessors: `tbl`, `pic`, `eqn`, and `refer`. You mark text meant for preprocessors by enclosing it in pairs of tags as follows.

.TS [H]

.TE

Denotes a table, to be processed by the `tbl` preprocessor. The optional argument `H` to `TS` instructs `groff` to create a running header with the information up to the `TH` macro. `groff` prints the header at the beginning of the table; if the table runs onto another page, `groff` prints the header on the next page as well.

.PS

.PE

Denotes a graphic, to be processed by the `pic` preprocessor. You can create a `pic` file by hand, using the AT&T `pic` manual available on the Web as a reference, or by using a graphics program such as `xfig`.

.EQ [*align*]

.EN

Denotes an equation, to be processed by the `eqn` preprocessor. The optional *align* argument can be `C`, `L`, or `I` to center (the default), left-justify, or indent the equation.

. [

.]

Denotes a reference, to be processed by the `refer` preprocessor. The GNU *refer(1)* man page provides a comprehensive reference to the preprocessor and the format of the bibliographic database.

4.3.5.9. An example multi-page table

The following is an example of how to set up a table that may print across two or more pages.

```
.TS H
allbox expand;
cb | cb .
Text      ...of heading...
—
.TH
.T&
l | l .
... the rest of the table follows...
.CW
.TE
```

4.3.5.10. Footnotes

The `ms` macro package has a flexible footnote system. You can specify either numbered footnotes or symbolic footnotes (that is, using a marker such as a dagger symbol).

`*[*]`

Specifies the location of a numbered footnote marker in the text.

`.FS`

`.FE`

Specifies the text of the footnote. The default action is to create a numbered footnote; you can create a symbolic footnote by specifying a *mark* glyph (such as `\[dg]` for the dagger glyph) in the body text and as an argument to the `FS` macro, followed by the text of the footnote and the `FE` macro.

You can control how `groff` prints footnote numbers by changing the value of the `FF` register. See [Document control registers](#).

Footnotes can be safely used within keeps and displays, but you should avoid using numbered footnotes within floating keeps. You can set a second `**` marker between a `**` and its corresponding `.FS` entry; as long as each `FS` macro occurs *after* the corresponding `**` and the occurrences of `.FS` are in the same order as the corresponding occurrences of `**`.

4.3.6. Page layout

The default output from the `ms` macros provides a minimalist page layout: it prints a single column, with the page number centered at the top of each page. It prints no footers.

You can change the layout by setting the proper number registers and strings.

4.3.6.1. Headers and footers

For documents that do not distinguish between odd and even pages, set the following strings:

`*[LH]`

`*[CH]`

`*[RH]`

Sets the left, center, and right headers.

* [LF]

* [CF]

* [RF]

Sets the left, center, and right footers.

For documents that need different information printed in the even and odd pages, use the following macros:

.OH 'left' center' right'

.EH 'left' center' right'

.OF 'left' center' right'

.EF 'left' center' right'

The OH and EH macros define headers for the odd and even pages; the OF and EF macros define footers for the odd and even pages. This is more flexible than defining the individual strings.

You can replace the quote (') marks with any character not appearing in the header or footer text.

To specify custom header and footer processing, redefine the following macros:

.PT

.HD

.BT

The PT macro defines a custom header; the BT macro defines a custom footer. These macros must handle odd/even/first page differences if necessary.

The HD macro defines additional header processing to take place after executing the PT macro.

4.3.6.2. Margins

You control margins using a set of number registers. See [Document control registers](#), for details.

4.3.6.3. Multiple columns

The mS macros can set text in as many columns as do reasonably fit on the page. The following macros are available; all of them force a page break if a multi-column mode is already set. However, if the current mode is single-column, starting a multi-column mode does *not* force a page break.

.1C

Single-column mode.

.2C

Two-column mode.

.MC [*width* [*gutter*]]

Multi-column mode. If you specify no arguments, it is equivalent to the 2C macro. Otherwise, *width* is the width of each column and *gutter* is the space between columns. The MINGW n umber register controls the default gutter width.

4.3.6.4. Creating a table of contents

The facilities in the `ms` macro package for creating a table of contents are semi-automated at best. Assuming that you want the table of contents to consist of the document's headings, you need to repeat those headings wrapped in `XS` and `XE` macros.

```
.XS [page]
.XA [page]
.XE
```

These macros define a table of contents or an individual entry in the table of contents, depending on their use. The macros are very simple; they cannot indent a heading based on its level. The easiest way to work around this is to add tabs to the table of contents string. The following is an example:

```
.NH 1
Introduction
.XS
Introduction
.XE
.LP
...
.CW
.NH 2
Methodology
.XS
Methodology
.XE
.LP
...
```

You can manually create a table of contents by beginning with the `XS` macro for the first entry, specifying the page number for that entry as the argument to `XS`. Add subsequent entries using the `XA` macro, specifying the page number for that entry as the argument to `XA`. The following is an example:

```
.XS 1
Introduction
.XA 2
A Brief History of the Universe
.XA 729
Details of Galactic Formation
...
.XE
```

```
.TC [no]
```

Prints the table of contents on a new page, setting the page number to `i` (Roman lowercase numeral one). You should usually place this macro at the end of the file, since `groff` is a single-pass formatter and can only print what has been collected up

to the point that the `TC` macro appears.

The optional argument `no` suppresses printing the title specified by the string register `TOC`.

`.PX [no]`

Prints the table of contents on a new page, using the current page numbering sequence. Use this macro to print a manually generated table of contents at the beginning of your document.

The optional argument `no` suppresses printing the title specified by the string register `TOC`.

The *Groff and Friends HOWTO* includes a `sed` script that automatically inserts `XS` and `XE` macro entries after each heading in a document.

Altering the `NH` macro to automatically build the table of contents is perhaps initially more difficult, but would save a great deal of time in the long run if you use `ms` regularly.

4.3.6.5. Strings and Special Characters

The `ms` macros provide the following predefined strings. You can change the string definitions to help in creating documents in languages other than English.

`* [REFERENCES]`

Contains the string printed at the beginning of the references (bibliography) page. The default is 'References'.

`* [ABSTRACT]`

Contains the string printed at the beginning of the abstract. The default is 'ABSTRACT'.

`* [TOC]`

Contains the string printed at the beginning of the table of contents.

`* [MONTH1]`

`* [MONTH2]`

`* [MONTH3]`

`* [MONTH4]`

`* [MONTH5]`

`* [MONTH6]`

`* [MONTH7]`

`* [MONTH8]`

`* [MONTH9]`

`* [MONTH10]`

`* [MONTH11]`

`* [MONTH12]`

Prints the full name of the month in dates. The default is 'January', 'February', etc.

The following special characters are available⁸ : " -- Special Characters

`* [-]`

Prints an em dash.

`* [Q]`

`* [U]`

⁸ For an explanation what special characters are see [Special Characters](#).

Prints typographer's quotes in troff, and plain quotes in nroff. `*Q` is the left quote and `*U` is the right quote.

Improved accent marks are available in the `ms` macros.

`.AM`

Specify this macro at the beginning of your document to enable extended accent marks and special characters. This is a Berkeley extension.

To use the accent marks, place them **after** the character being accented.

Note that groff's native support for accents is superior to the following definitions.

The following accent marks are available after invoking the `AM` macro:

`*[']`

Acute accent.

`*[`]`

Grave accent.

`*[^]`

Circumflex.

`*[,]`

Cedilla.

`*[~]`

Tilde.

Umlaut.

`*[ˇ]`

Hacek.

`*[_]`

Macron (overbar).

`*[.]`

Underdot.

`*[o]`

Ring above.

The following are standalone characters available after invoking the `AM` macro:

`*[?]`

Upside-down question mark.

`*[!]`

Upside-down exclamation point.

`*[8]`

German ß ligature.

`*[3]`

Yogh.

`*[Th]`

Uppercase thorn.

`*[th]`

Lowercase thorn.

`*[D-]`

Uppercase eth.

`*[d-]`

Lowercase eth.

`*[q]`

Hooked o.

`*[ae]`

Lowercase æ ligature.

`*[Ae]`

Uppercase Æ ligature.

4.3.7. Differences from AT&T `ms`

This section lists the (minor) differences between the `groff -ms` macros and AT&T `troff -ms` macros.

- The internals of `groff -ms` differ from the internals of AT&T `troff -ms`. Documents that depend upon implementation details of AT&T `troff -ms` may not format properly with `groff -ms`.
- The general error-handling policy of `groff -ms` is to detect and report errors, rather than silently to ignore them.
- `groff -ms` does not work in compatibility mode (that is, with the `-C` option).
- There is no special support for typewriter-like devices.
- `groff -ms` does not provide cut marks.
- Multiple line spacing is not supported. Use a larger vertical spacing instead.
- Some UNIX `ms` documentation says that the `CW` and `GW` number registers can be used to control the column width and gutter width, respectively. These number registers are not used in `groff -ms`.
- Macros that cause a reset (paragraphs, headings, etc.) may change the indentation. Macros that change the indentation do not increment or decrement the indentation, but rather set it absolutely. This can cause problems for documents that define additional macros of their own. The solution is to use not the `in` request but instead the `RS` and `RE` macros.
- 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 evaluated. This implies that `PO` should not be used early in the document, unless it is changed also: Remember that accessing an undefined register automatically defines it.

`\n[GS]`

This number register is set to 1 by the `groff -ms` macros, but it is not used by the AT&T `troff -ms` macros. Documents that need to determine whether they are being formatted with AT&T `troff -ms` or `groff -ms` should use this number register.

Emulations of a few ancient Bell Labs macros can be re-enabled by calling the otherwise undocumented `SC` section-header macro. Calling `SC` enables `UC` for marking up a product or application name, and the pair `P1/P2` for surrounding code example displays.

These are not enabled by default because (a) they were not documented, in the original `ms` manual, and (b) the `P1` and `UC` macros collide with different macros with the same names in the Berkeley version of `ms`.

These `groff` emulations are sufficient to give back the 1976 Kernighan & Cherry paper *Typesetting Mathematics -- User's Guide* its section headings, and restore some text that had gone missing as arguments of undefined macros. No warranty express or implied is given as to how well the typographic details these produce match the original Bell Labs macros.

4.3.7.1. `troff` macros not appearing in `groff`

Macros missing from `groff -ms` are cover page macros specific to Bell Labs and Berkeley. The macros known to be missing are:

<code>.TM</code>	Technical memorandum; a cover sheet style
<code>.IM</code>	Internal memorandum; a cover sheet style
<code>.MR</code>	Memo for record; a cover sheet style
<code>.MF</code>	Memo for file; a cover sheet style
<code>.EG</code>	Engineer's notes; a cover sheet style
<code>.TR</code>	Computing Science Tech Report; a cover sheet style
<code>.OK</code>	Other keywords
<code>.CS</code>	Cover sheet information
<code>.MH</code>	A cover sheet macro

4.3.7.2. `groff` macros not appearing in AT&T `troff`

The `groff -ms` macros have a few minor extensions compared to the AT&T `troff -ms` macros.

<code>.AM</code>	Improved accent marks. See Strings and Special Characters , for details.
<code>.DS I</code>	Indented display. The default behavior of AT&T <code>troff -ms</code> was to indent; the <code>groff</code> default prints displays flush left with the body text.
<code>.CW</code>	Print text in <code>constant width</code> (Courier) font.
<code>.IX</code>	Indexing term (printed on standard error). You can write a script to capture and process an index generated in this manner.

The following additional number registers appear in `groff -ms`:

`\n[MINGW]`

Specifies a minimum space between columns (for multi-column output); this takes the place of the `GW` register that was documented but apparently not implemented in AT&T `troff`.

Several new string registers are available as well. You can change these to handle (for example) the local language. See [Strings and Special Characters](#), for details.

4.3.8. Naming Conventions

The following conventions are used for names of macros, strings and number registers. External names available to documents that use the `groff -ms` macros contain only uppercase letters and digits.

Internally the macros are divided into modules; naming conventions are 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.
- Constructed names used to implement arrays are of the form *array!index*.

Thus the `groff ms` macros reserve the following names:

- Names containing the characters `*`, `@`, and `:`.
- Names containing only uppercase letters and digits.

4.4. `me`

See the `meintro.me` and `mereref.me` documents in `groff's doc` directory.

4.5. `mm`

NAME

`groff_mm` – memorandum macros for GNU `roff`

SYNOPSIS

groff -mm [*options...*] [*files...*]

DESCRIPTION

The `groff mm` macros are intended to be compatible with the DWB `mm` macros with the following limitations:

- No Bell Labs localisms are implemented.
- The macros `OK` and `PM` are not implemented.

- groff mm does not support cut marks.

mm is intended to support easy localization. Use **mmse** as an example how to adapt the output format to a national standard. Localized strings are collected in the file `'/usr/share/groff/1.22.3.rc1.24-ea225/tmac/xx.tmac'`, where *xx* denotes the two-letter code for the *language*, as defined in the ISO 639 standard. For Swedish, this is `'sv.tmac'` – not `'se'`, which is the ISO 3166 two-letter code for the *country* (as used for the output format localization).

A file called **locale** or *country_locale* is read after the initialization of the global variables. It is therefore possible to localize the macros with a different company name and so on.

In this manual, square brackets are used to show optional arguments.

Number registers and strings

Many macros can be controlled by number registers and strings. A number register is assigned with the **nr** command:

```
.nr XXX [ $\pm$ ]n [i]/R
```

XXX is the name of the register, *n* is the value to be assigned, and *i* is the increment value for auto-increment. *n* can have a plus or minus sign as a prefix if an increment or decrement of the current value is wanted. (Auto-increment or auto-decrement occurs if the number register is used with a plus or minus sign, `\nXXX` or `\n-[XXX]`.)

Strings are defined with **ds**.

```
.ds YYY string
```

The string is assigned everything to the end of the line, even blanks. Initial blanks in *string* should be prefixed with a double-quote. (Strings are used in the text as `*[YYY]`.)

Special formatting of number registers

A number register is printed with normal digits if no format has been given. Set the format with **af**:

```
.af R c
```

R is the name of the register, *c* is the format.

Form	Sequence
1	0, 1, 2, 3, ...
001	000, 001, 002, 003, ...
<i>i</i>	0, i, ii, iii, iv, ...
I	0, I, II, III, IV, ...
<i>a</i>	0, a, b, c, ..., z, aa, ab, ...
A	0, A, B, C, ..., Z, AA, AB, ...

Fonts

In **mm**, the fonts (or rather, font styles) **R** (normal), **I** (italic), and **B** (bold) are hard-wired to font positions **1**, **2**, and **3**, respectively. Internally, font positions are used for backwards compatibility. From a practical point of view it doesn't make a big difference – a different font family can still be selected with a call to the **.fam** request or

using **groff's** **-f** command-line option. On the other hand, if you want to replace just, say, font **B**, you have to replace the font at position 2 (with a call to `'fp 2 . . .'`).

Macros

)E *level text*

Add heading text *text* to the table of contents with *level*, which is either 0 or in the range 1 to 7. See also **.H**. This macro is used for customized tables of contents.

1C [1]

Begin one-column processing. A **1** as an argument disables the page break. Use wide footnotes, small footnotes may be overprinted.

2C Begin two-column processing. Splits the page in two columns. It is a special case of **MC**. See also **1C**.

AE Abstract end, see **AS**.

AF [*name-of-firm*]

Author's firm, should be called before **AU**, see also **COVER**.

AL [*type* [*text-indent* [1]]]

Start auto-increment list. Items are numbered beginning with one. The *type* argument controls the format of numbers.

Arg	Description
1	Arabic (the default)
A	Upper-case letters (A–Z)
a	Lower-case letters (a–z)
l	Upper-case roman
i	Lower-case roman

text-indent sets the indentation and overrides **Li**. A third argument prohibits printing of a blank line before each item.

APP *name text*

Begin an appendix with name *name*. Automatic naming occurs if *name* is `""`. The appendices start with **A** if automatic naming is used. A new page is ejected, and a header is also produced if the number variable **Aph** is non-zero. This is the default. The appendix always appears in the 'List of contents' with correct page numbers. The name 'APPENDIX' can be changed by setting the string **App** to the desired text. The string **Apptxt** contains the current appendix text.

APPSK *name pages text*

Same as **.APP**, but the page number is incremented with *pages*. This is used when diagrams or other non-formatted documents are included as appendices.

AS [*arg* [*indent*]]

Abstract start. Indentation is specified in 'ens', but scaling is allowed. Argument *arg* controls where the abstract is printed.

Arg	Placement
------------	------------------

BVL *text-indent* [*mark-indent* [1]]

Start of broken variable-item list. Broken variable-item list has no fixed mark, it assumes that every **LI** has a mark instead. The text always begins at the next line after the mark. *text-indent* sets the indentation to the text, and *mark-indent* the distance from the current indentation to the mark. A third argument prohibits printing of a blank line before each item.

COVER [*arg*]

Begin a coversheet definition. It is important that **.COVER** appears before any normal text. This macro uses *arg* to build the filename `'/usr/share/groff/1.22.3.rc1.24-ea225/tmac/mm/arg.cov'`. Therefore it is possible to create unlimited types of cover sheets. `'ms.cov'` is supposed to look like the ms cover sheet. **.COVER** requires a **.COVEND** at the end of the cover definition. Always use this order of the cover macros:

```
.COVER
.TL
.AF
.AU
.AT
.AS
.AE
.COVEND
```

However, only **.TL** and **.AU** are required.

COVEND

Finish the cover description and print the cover page. It is defined in the cover file.

DE Display end. Ends a block of text or display that begins with **DS** or **DF**.

DF [*format* [*fill* [*rindent*]]]

Begin floating display (no nesting allowed). A floating display is saved in a queue and is printed in the order entered. *Format*, *fill*, and *rindent* are the same as in **DS**. Floating displays are controlled by the two number registers **De** and **Df**.

De register

- 0 Nothing special, this is the default.
- 1 A page eject occurs after each printed display, giving only one display per page and no text following it.

Df register

- 0 Displays are printed at the end of each section (when section-page numbering is active) or at the end of the document.
- 1 A new display is printed on the current page if there is enough space, otherwise it is printed at the end of the document.
- 2 One display is printed at the top of each page or column (in multi-column mode).

- 3 Print one display if there is enough space for it, otherwise it is printed at the top of the next page or column.
- 4 Print as many displays as possible in a new page or column. A page break occurs between each display if **De** is not zero.
- 5 Fill the current page with displays and the rest beginning at a new page or column. (This is the default.) A page break occurs between each display if **De** is not zero.

DL [*text-indent* [**1** [**1**]]]

Dash list start. Begins a list where each item is printed after a dash. *text-indent* changes the default indentation of the list items set by number register **Pi**. A second argument prevents an empty line between each list item. See **LI**. A third argument prohibits printing of a blank line before each item.

DS [*format* [*fill* [*rindent*]]]

Static display start. Begins collection of text until **DE**. The text is printed together on the same page, unless it is longer than the height of the page. **DS** can be nested arbitrarily.

format

""	No indentation.
none	No indentation.
L	No indentation.
I	Indent text with the value of number register Si .
C	Center each line.
CB	Center the whole display as a block.
R	Right-adjust the lines.
RB	Right-adjust 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 reasons.

fill

""	Line-filling turned off.
none	Line-filling turned off.
N	Line-filling turned off.
F	Line-filling turned on.

'N' and 'F' can also be specified as '0' and '1', respectively.

By default, an empty line is printed before and after the display. Setting number register **Ds** to 0 prevents this. *rindent* shortens the line length by that amount.

EC [*title* [*override* [*flag* [*refname*]]]]

Equation title. Sets a title for an equation. The *override* argument changes the numbering.

flag

none	<i>override</i> is a prefix to the number.
0	<i>override</i> is a prefix to the number.

- 1 *override* is a suffix to the number.
- 2 *override* replaces the number.

EC uses the number register **Ec** as a counter. It is possible to use **.af** to change the format of the number. If number register **Of** is 1, the format of title uses a dash instead of a dot after the number.

The string **Le** controls the title of the List of Equations; default is 'LIST OF EQUATIONS'. The List of Equations is only printed if number register **Le** is 1. The default is 0. The string **Liec** contains the word 'Equation', which is printed before the number. If *refname* is used, then the equation number is saved with **.SETR**, and can be retrieved with '**.GETST refname**'.

Special handling of the title occurs if **EC** is used inside **DS/DE**; it is not affected by the format of **DS**.

EF [*arg*]

Even-page footer, printed just above the normal page footer on even pages. See **PF**.

This macro defines string **EOPef**.

EH [*arg*]

Even-page header, printed just below the normal page header on even pages. See **PH**.

This macro defines string **TPeh**.

EN Equation end, see **EQ**.

EOP

End-of-page user-defined macro. This macro is called instead of the normal printing of the footer. The macro is executed in a separate environment, without any trap active. See **TP**.

Strings available to EOP

EOPf	argument of PF
EOPef	argument of EF
EOPof	argument of OF

EPIC [**-L**] *width height* [*name*]

Draw a box with the given *width* and *height*. It also prints the text *name* or a default string if *name* is not specified. This is used to include external pictures; just give the size of the picture. **-L** left-adjusts the picture; the default is to center. See **PIC**.

EQ [*label*]

Equation start. **EQ/EN** are the delimiters for equations written for **eqn(1)**. **EQ/EN** must be inside of a **DS/DE** pair, except if **EQ** is used to set options for **eqn** only. The *label* argument appears at the right margin of the equation, centered vertically within the **DS/DE** block, unless number register **Eq** is 1. Then the label appears at the left margin.

If there are multiple **EQ/EN** blocks within a single **DS/DE** pair, only the last equation label (if any) is printed.

EX [*title* [*override* [*flag* [*refname*]]]]

Exhibit title. The arguments are the same as for **EC**. **EX** uses the number register **Ex** as a counter. The string **Lx** controls the title of the List of Exhibits; default is 'LIST OF EXHIBITS'. The List of Exhibits is only printed if number register **Lx** is 1, which is the default. The string **Lie x** contains the word 'Exhibit', which is printed before the number. If *refname* is used, the exhibit number is saved with **.SETR**, and can be retrieved with '**.GETST refname**'.

Special handling of the title occurs if **EX** is used inside **DS/DE**; it is not affected by the format of **DS**.

FC [*closing*]

Print 'Yours very truly,' as a formal closing of a letter or memorandum. The argument replaces the default string. The default is stored in string variable **Letfc**.

FD [*arg* [1]]

Footnote default format. Controls the hyphenation (hyphen), right margin justification (adjust), and indentation of footnote text (indent). It can also change the label justification (ljust).

arg	hyphen	adjust	indent	ljust
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 argument greater than or equal to 11 is considered as value 0. Default for **mm** is 10.

FE Footnote end.**FG** [*title* [*override* [*flag* [*refname*]]]]

Figure title. The arguments are the same as for **EC**. **FG** uses the number register **Fg** as a counter. The string **Lf** controls the title of the List of Figures; default is 'LIST OF FIGURES'. The List of Figures is only printed if number register **Lf** is 1, which is the default. The string **Lifg** contains the word 'Figure', which is printed before the number. If *refname* is used, then the figure number is saved with **.SETR**, and can be retrieved with '**.GETST refname**'.

Special handling of the title occurs if **FG** is used inside **DS/DE**, it is not affected by the format of **DS**.

FS [*label*]

Footnote start. The footnote is ended by **FE**. By default, footnotes are automatically numbered; the number is available in string **F**. Just add ***F** in the text.

By adding *label*, it is possible to have other number or names on the footnotes. Footnotes in displays are now possible. An empty line separates footnotes; the height of the line is controlled by number register **Fs**, default value is 1.

GETHN *refname* [*varname*]

Include the header number where the corresponding '**SETR** *refname*' was placed. This is displayed as 'X.X.X.' in pass 1. See **INITR**. If *varname* is used, **GETHN** sets the string variable *varname* to the header number.

GETPN *refname* [*varname*]

Include the page number where the corresponding '**SETR** *refname*' was placed. This is displayed as '9999' in pass 1. See **INITR**. If *varname* is used, **GETPN** sets the stringvariable *varname* to the page number.

GETR *refname*

Combine **GETHN** and **GETPN** with the text 'chapter' and ', page'. The string **Qrf** contains the text for the cross reference:

.ds Qrf See chapter *[Qrfh], page *[Qrfp].

Qrf may be changed to support other languages. Strings **Qrfh** and **Qrfp** are set by **GETR** and contain the page and header number, respectively.

GETST *refname* [*varname*]

Include the string saved with the second argument to **SETR**. This is a dummy string in pass 1. If *varname* is used, **GETST** sets it to the saved string. See **INITR**.

H *level* [*heading-text* [*heading-suffix*]]

Numbered section heading. Section headers can have a level between 1 and 14; level 1 is the top level. The text is given in *heading-text*, and must be surrounded by double quotes if it contains spaces. *heading-suffix* is added to the header in the text but not in the table of contents. This is normally used for footnote marks and similar things. Don't use ***F** in *heading-suffix*, it doesn't work. A manual label must be used, see **FS**. A call to the paragraph macro **P** directly after **H** is ignored. **H** takes care of spacing and indentation.

Page ejection before heading

Number register **Ej** controls page ejection before the heading. By default, a level-one heading gets two blank lines before it; higher levels only get one. A new page is ejected before each first-level heading if number register **Ej** is 1. All levels below or equal the value of **Ej** get a new page. Default value for **Ej** is 0.

Heading break level

A line break occurs after the heading if the heading level is less or equal to number register **Hb**. Default value is 2.

Heading space level

A blank line is inserted after the heading if the heading level is less or equal to number register **Hs**. Default value is 2.

Text follows the heading on the same line if the level is greater than both **Hb** and **Hs**.

Post-heading indent

Indentation of the text after the heading is controlled by number register **Hi**. Default value is 0.

Hi

- 0 The text is left-justified.
- 1 Indentation of the text follows the value of number register **Pt**, see **P**.
- 2 The text is lined up with the first word of the heading.

Centered section headings

All headings whose level is equal or below number register **Hc** and also less than or equal to **Hb** or **Hs** are centered.

Font control of the heading

The font of each heading level is controlled by string **HF**. It contains a font number or font name for each level. Default value is

2 2 2 2 2 2 2 2 2 2 2 2 2 2

(all headings in italic). This could also be written as

IIIIIIIIIIIIIIIIIIII

Note that some other implementations use **3 3 2 2 2 2 2** as the default value. All omitted values are presumed to have value 1.

Point size control

String **HP** controls the point size of each heading, in the same way as **HF** controls the font. A value of 0 selects the default point size. Default value is

0 0 0 0 0 0 0 0 0 0 0 0 0 0

Beware that only the point size changes, not the vertical size. The latter can be controlled by the user-specified macros **HX** and/or **HZ**.

Heading counters

Fourteen number registers named **H1** up to **H14** contain the counter for each heading level. The values are printed using Arabic numerals; this can be changed with the macro **HM** (see below). All marks are concatenated before printing. To avoid this, set number register **Ht** to 1. This only prints the current heading counter at each heading.

Automatic table of contents

All headings whose level is equal or below number register **CI** are saved to be printed in the table of contents. Default value is 2.

Special control of the heading, user-defined macros

The following macros can be defined by the user to get a finer control of vertical spacing, fonts, or other features. Argument *level* is the level-argument to **H**, but 0 for unnumbered headings (see **HU**). Argument *real level* is the real level; it is set to number register **Hu** for unnumbered headings. Argument *heading-text* is the text argument to **H** and **HU**.

HX *level rlevel heading-text*

This macro is called just before the printing of the heading. The following registers are available for **HX**. Note that **HX** may alter **}0**, **}2**, and **;3**.

}0 (string)

Contains the heading mark plus two spaces if *rlevel* is non-zero, otherwise empty.

;0 (register)

Contains the position of the text after the heading. 0 means that the text should follow the heading on the same line, 1 means that a line break should occur before the text, and 2 means that a blank line should separate the heading and the text.

}2 (string)

Contains two spaces if register **;0** is 0. It is used to separate the heading from the text. The string is empty if **;0** is non-zero.

;3 (register)

Contains the needed space in units after the heading. Default is 2v. Can be used to change things like numbering (**}0**), vertical spacing (**}2**), and the needed space after the heading.

HY *dlevel rlevel heading-text*

This macro is called after size and font calculations and might be used to change indentation.

HZ *dlevel rlevel heading-text*

This macro is called after the printing of the heading, just before **H** or **HU** exits. Can be used to change the page header according to the section heading.

HC [*hyphenation-character*]

Set hyphenation character. Default value is '%'. Resets to the default if called without argument. Hyphenation can be turned off by setting number register **Hy** to 0 at the beginning of the file.

HM [*arg1 [arg2 [... [arg14]]]]*]

Heading mark style. Controls the type of marking for printing of the heading counters. Default is 1 for all levels.

Argument

1	Arabic numerals.
0001	Arabic numerals with leading zeroes, one or more.
A	upper-case alphabetic
a	lower-case alphabetic
I	upper-case roman numerals
i	lower-case roman numerals
""	Arabic numerals.

HU *heading-text*

Unnumbered section header. **HU** behaves like **H** at the level in number register

Hu. See**H**.

HX *dlevel rlevel heading-text*

User-defined heading exit. Called just before printing the header. See**H**.

HY *dlevel rlevel heading-text*

User-defined heading exit. Called just before printing the header. See**H**.

HZ *dlevel rlevel heading-text*

User-defined heading exit. Called just after printing the header. See**H**.

I [*italic-text* [*prev-font-text* [*italic-text* [. . .]]]]

Italic. Changes the font to italic if called without arguments. With one argument it sets the word in italic. With two arguments it concatenates them and sets the first word in italic and the second in the previous font. There is no limit on the number of argument; all are concatenated.

IA [*addressee-name* [*title*]]

Begin specification of the addressee and addressee's address in letter style. Several names can be specified with empty **IA/IE**-pairs, but only one address. See **LT**.

IB [*italic-text* [*bold-text* [*italic-text* [. . .]]]]

Italic-bold. Even arguments are printed in italic, odd in boldface. See**I**.

IE End the address specification after **IA**.

INITI *type filename [macro]*

Initialize the new index system and set the filename to collect index lines in with **IND**. Argument *type* selects the type of index: page number, header marks or both. The default is page numbers.

It is also possible to create a macro that is responsible for formatting each row; just add the name of the macro as a third argument. The macro is then called with the index as argument(s).

type

N Page numbers

H Header marks

B Both page numbers and header marks, separated with a tab character.

INITR *filename*

Initialize the cross reference macros. Cross references are written to stderr and are supposed to be redirected into file '*filename.qrf*'. Requires two passes with groff; this is handled by a separate program called **mmroff**(1). This program exists because **groff**(1) by default deactivates the unsafe operations that are required by **INITR**. The first pass looks for cross references, and the second one includes them. **INITR** can be used several times, but it is only the first occurrence of **INITR** that is active.

See also **SETR**, **GETPN**, and **GETHN**.

IND *arg1 [arg2 [. . .]]*

Write a line in the index file selected by **INITI** with all arguments and the page number or header mark separated by tabs.

Examples

```
arg1\tpage number
arg1\targ2\tpage number
arg1\thead mark
arg1\tpage number\thead mark
```

INDP

Print the index by running the command specified by string variable **Indcmd**, which has 'sort -t\t' as the default value. **INDP** reads the output from the command to form the index, by default in two columns (this can be changed by defining **TYIND**). The index is printed with string variable **Index** as header, default is 'INDEX'. One-column processing is reactivated after the list. **INDP** calls the user-defined macros **TXIND**, **TYIND**, and **TZIND** if defined. **TXIND** is called before printing the string 'INDEX', **TYIND** is called instead of printing 'INDEX', and **TZIND** is called after the printing and should take care of restoring to normal operation again.

ISODATE[0]

Change the predefined date string in **DT** to ISO-format, this is, 'YYYY-MM-DD'. This can also be done by adding **-rIso=1** on the command line. Reverts to old date format if argument is **0**.

IR [*italic-text* [*roman-text* [*italic-text* [. . .]]]]

Italic-roman. Even arguments are printed in italic, odd in roman. See **I**.

LB *text-indent* *mark-indent* *pad* *type* [*mark* [*LI-space* [*LB-space*]]]

List-begin macro. This is the common macro used for all lists. *text-indent* is the number of spaces to indent the text from the current indentation.

pad and *mark-indent* control where to put the mark. The mark is placed within the mark area, and *mark-indent* sets the number of spaces before this area. By default it is 0. The mark area ends where the text begins. The start of the text is still controlled by *text-indent*.

The mark is left-justified within the mark area if *pad* is 0. If *pad* is greater than 0, *mark-indent* is ignored, and the mark is placed *pad* spaces before the text. This right-justifies the mark.

If *type* is 0 the list either has a hanging indentation or, if argument *mark* is given, the string *mark* as a mark.

If *type* is greater than 0 automatic numbering occurs, using arabic numbers if *mark* is empty. *mark* can then be any of '1', 'A', 'a', 'l', or 'i'.

type selects one of six possible ways to display the mark.

type

- | | |
|---|-----|
| 1 | x. |
| 2 | x) |
| 3 | (x) |
| 4 | [x] |
| 5 | <x> |
| 6 | {x} |

Every item in the list gets *LI-space* number of blank lines before them. Default is 1.

LB itself prints *LB-space* blank lines. Default is 0.

LC [*list-level*]

List-status clear. Terminates all current active lists down to *list-level*, or 0 if no argument is given. This is used by **H** to clear any active list.

LE[1]

List end. Terminates the current list. **LE** outputs a blank line if an argument is given.

LI [*mark* [1[2]]]

List item preceding every item in a list. Without argument, **LI** prints the mark determined by the current list type. By giving **LI** one argument, it uses that as the mark instead. Two arguments to **LI** makes *mark* a prefix to the current mark. There is no separating space between the prefix and the mark if the second argument is '2' instead of '1'. This behaviour can also be achieved by setting number register **Limsp** to zero. A zero length *mark* makes a hanging indentation instead.

A blank line is printed before the list item by default. This behaviour can be controlled by number register **Ls**. Pre-spacing occurs for each list level less than or equal to **Ls**. Default value is 99. There is no nesting limit.

The indentation can be changed through number register **Li**. Default is 6.

All lists begin with a list initialization macro, **LB**. There are, however, seven predefined list types to make lists easier to use. They all call **LB** with different default values.

AL	Automatically Incremented List
ML	Marked List
VL	Variable-Item List
BL	Bullet List
DL	Dash List
RL	Reference List
BVL	Broken Variable List.

These lists are described at other places in this manual. See also **LB**.

LT [*arg*]

Format a letter in one of four different styles depending on the argument. See also section **INTERNALS**.

Arg	Style
BL	Blocked. Date line, return address, writer's address and closing begins at the center of the line. All other lines begin at the left margin.
SB	Semi-blocked. Same as blocked, except that the first line in every paragraph is indented five spaces.
FB	Full-blocked. All lines begin at the left margin.

- SP Simplified. Almost the same as the full-blocked style. Subject and the writer's identification are printed in all-capital.

LO *type* [*arg*]

Specify options in letter (see **.LT**). This is a list of the standard options:

- CN Confidential notation. Prints 'CONFIDENTIAL' on the second line below the date line. Any argument replaces 'CONFIDENTIAL'. See also string variable **LetCN**.
- RN Reference notation. Prints 'In reference to:' and the argument two lines below the date line. See also string variable **LetRN**.
- AT Attention. Prints 'ATTENTION:' and the argument below the inside address. See also string variable **LetAT**.
- SA Salutation. Prints 'To Whom It May Concern:' or the argument if it was present. The salutation is printed two lines below the inside address. See also string variable **LetSA**.
- SJ Subject line. Prints the argument as subject prefixed with 'SUBJECT:' two lines below the inside address, except in letter type 'SP', where the subject is printed in all-capital without any prefix. See also string variable **LetSJ**.

MC *column-size* [*column-separation*]

Begin multiple columns. Return to normal with **1C**. **MC** creates as many columns as the current line length permits. *column-size* is the width of each column, and *column-separation* is the space between two columns. Default separation is *column-size*/15. See also **1C**.

ML *mark* [*text-indent* [**1**]]

Marked list start. The *mark* argument is printed before each list item. *text-indent* sets the indent and overrides **Li**. A third argument prohibits printing of a blank line before each item.

MT [*arg* [*addressee*]]

Memorandum type. The argument *arg* is part of a filename in '/usr/share/groff/1.22.3.rc1.24-*ea225*/tmac/mm/**.MT*'. Memorandum types 0 to 5 are supported, including type 'string' (which gets internally mapped to type 6). *addressee* just sets a variable, used in the AT&T macros.

arg

- 0 Normal memorandum, no type printed.
- 1 Memorandum with 'MEMORANDUM FOR FILE' printed.
- 2 Memorandum with 'PROGRAMMER'S NOTES' printed.
- 3 Memorandum with 'ENGINEER'S NOTES' printed.
- 4 Released paper style.
- 5 External letter style.

See also **COVER/COVEND**, a more flexible type of front page.

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 space1 [cw2 space2 [cw3 . . .]]*

Begin a special multi-column mode. All columns widths must be specified. The space between the columns must be specified also. The last column does not need any space definition. **MULB** starts a diversion, and **MULE** ends the diversion and prints the columns. The unit for the width and space arguments is 'n', but **MULB** accepts all normal unit specifications like 'c' and 'i'. **MULB** operates in a separate environment.

MULN

Begin the next column. This is the only way to switch the column.

MULE

End the multi-column mode and print the columns.

nP [*type*]

Print numbered paragraph with header level two. See **P**.

NCOL

Force printing to the next column. Don't use this together with the **MUL*** macros, see **2C**.

NS [*arg* [**1**]]

Print different types of notations. The argument selects between the predefined type of notations. If the second argument is available, then the argument becomes the entire notation. If the argument doesn't select a predefined type, it is printed as 'Copy (*arg*) to'. It is possible to add more standard notations, see the string variables **Letns** and **Letnsdef**.

Arg	Notation
<i>none</i>	Copy To
""	Copy To
1	Copy To (with att.) to
2	Copy To (without att.) to
3	Att.
4	Atts.
5	Enc.
6	Encs.
7	Under separate cover
8	Letter to
9	Memorandum to
10	Copy (with atts.) to
11	Copy (without atts.) to
12	Abstract Only to
13	Complete Memorandum to
14	CC

ND *new-date*

New date. Overrides the current date. Date is not printed if *new-date* is an empty string.

OF [*arg*]

Odd-page footer, a line printed just above the normal footer. See **EF** and **PF**.

This macro defines string **EOPof**.

OH [*arg*]

Odd-page header, a line printed just below the normal header. See **EH** and **PH**.

This macro defines string **TPoh**.

OP Make sure that the following text is printed at the top of an odd-numbered page. Does not output an empty page if currently at the top of an odd page.

P [*type*]

Begin new paragraph. **P** without argument produces left-justified text, even the first line of the paragraph. This is the same as setting *type* to 0. If the argument is 1, the first line of text following **P** is indented by the number of spaces in number register **Pi**, by default 5.

Instead of giving an argument to **P** it is possible to set the paragraph type in number register **Pt**. Using 0 and 1 is the same as adding that value to **P**. A value of 2 indents all paragraphs, except after headings, lists, and displays (this value can't be used as an argument to **P** itself).

The space between two paragraphs is controlled by number register **Ps**, and is 1 by default (one blank line).

PGFORM [*linelength* [*pagelength* [*pageoffset* [**1**]]]]

Set line length, page length, and/or page offset. This macro can be used for special formatting, like letter heads and other. It is normally the first command 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

No header is printed on the next page. Used to get rid of the header in letters or other special texts. This macro must be used before any text to inhibit the page header on the first page.

PIC [**-B**] [**-L**] [**-C**] [**-R**] [**-I** *n*] *filename* [*width* [*height*]]

Include a PostScript file in the document. The macro depends on **mmroff**(1) and **INITR**. The arguments **-L**, **-C**, **-R**, and **-I** *n* adjust the picture or indent it. With no flag the picture is adjusted to the left. Adding **-B** draws a box around the picture. The optional *width* and *height* can also be given to resize the picture.

PE Picture end. Ends a picture for **pic**(@MAN1EXT).

PF [*arg*]

Page footer. **PF** sets the line to be printed at the bottom of each page. Empty by default. See **PH** for the argument specification.

This macro defines string **EOPf**.

PH [*arg*]

Page header, a line printed at the top of each page. The argument should be specified as

"left-part center-part right-part"

where *left-part*, *center-part*, and *right-part* are printed left-justified, centered, and right justified, respectively. Within the argument to **PH**, the character '%' is changed to the current page number. The default argument is

"- % -"

which gives the page number between two dashes.

This macro defines string **TPh**.

- PS** Picture start (from pic). Begins a picture for **pic**(1).
- PX** Page header user-defined exit. This macro is called just after the printing of the page header in *no-space* mode.
- R** Roman. Return to roman font, see also **I**.
- RB** [*roman-text* [*bold-text* [*roman-text* [. . .]]]]
Roman-bold. Even arguments are printed in roman, odd in boldface. See **I**.
- RD** [*prompt* [*diversion* [*string*]]]
Read from standard input to diversion and/or string. The text is saved in a diversion named *diversion*. Recall the text by writing the name of the diversion after a dot on an empty line. A string is also defined if *string* is given. *Diversion* and/or *prompt* can be empty ("").
- RF** Reference end. Ends a reference definition and returns to normal processing. See **RS**.
- RI** [*roman-text* [*italic-text* [*roman-text* [. . .]]]]
Print even arguments in roman, odd in italic. See **I**.
- RL** [*text-indent*[1]]
Reference list start. Begins a list where each item is preceded with an automatically incremented number between square brackets. *text-indent* changes the default indentation.
- RP** [*arg1* [*arg2*]]
Produce reference page. This macro can be used if a reference page is wanted somewhere in the document. It is not needed if **TC** is used to produce a table of contents. The reference page is then printed automatically.
The reference counter is not reset if *arg1* is 1.
arg2 tells **RP** whether to eject a page or not.
- arg2**
- 0 The reference page is printed on a separate page.
 - 1 Do not eject page after the list.
 - 2 Do not eject page before the list.
 - 3 Do not eject page before and after the list.
- The reference items are separated by a blank line. Setting number register **Ls** to 0 suppresses the line.
- The string **Rp** contains the reference page title and is set to 'REFERENCES' by default. The number register **Rpe** holds the default value for the second argument of **RP**; it is initially set to 0.

RS [*string-name*]

Begin an automatically numbered reference definition. Put the string `*(Rf` where the reference mark should be and write the reference between **RS/RF** at next new line after the reference mark. The reference number is stored in number register **:R**. If *string-name* is given, a string with that name is defined and contains the current reference mark. The string can be referenced as `*[string-name]` later in the text.

S [*size* [*spacing*]]

Set point size and vertical spacing. If any argument is equal to 'P', the previous value is used. A 'C' means current value, and 'D' the default value. If '+' or '-' is used before the value, the current value is incremented or decremented, respectively.

SA [*arg*]

Set right-margin justification. Justification is turned on by default. No argument or value '0' turns off justification, and '1' turns on justification.

SETR *refname* [*string*]

Remember the current header and page number as *refname*. Saves *string* if *string* is defined. *string* is retrieved with **.GETST**. See **INITR**.

SG [*arg* [**1**]]

Signature line. Prints the authors name(s) after the formal closing. The argument is appended to the reference data, printed at either the first or last author. The reference data is the location, department, and initials specified with **.AU**. It is printed at the first author if the second argument is given, otherwise at the last. No reference data is printed if the author(s) is specified through **.WA/WE**. See section **INTERNALS**.

SK [*pages*]

Skip pages. If *pages* is 0 or omitted, a skip to the next page occurs unless it is already at the top of a page. Otherwise it skips *pages* pages.

SM *string1* [*string2* [*string3*]]

Make a string smaller. If *string2* is given, *string1* is made smaller and *string2* stays at normal size, concatenated with *string1*. With three arguments, everything is concatenated, but only *string2* is made smaller.

SP [*lines*]

Space vertically. *lines* can have any scaling factor, like '3i' or '8v'. Several **SP** calls in a line only produces the maximum number of lines, not the sum. **SP** is ignored also until the first text line in a page. Add **&** before a call to **SP** to avoid this.

TAB Reset tabs to every 5n. Normally used to reset any previous tab positions.

TB [*title* [*override* [*flag* [*refname*]]]]

Table title. The arguments are the same as for **EC**. **TB** uses the number register **Tb** as a counter. The string **Lt** controls the title of the List of Tables; default value is 'LIST OF TABLES'. The List of Tables is only printed if number register **Lt** is 1, which is the default. The string **Litb** contains the word 'TABLE', which is printed before the number.

Special handling of the title occurs if **TB** is used inside **DS/DE**, it is not affected by the format of **DS**.

TPoh argument of **OH**

TS[H]

Table start. This is the start of a table specification to **tbl**(1). **TS** ends with **TE**. Argument 'H' tells **mm** that the table has a header. See **TH**.

TX User-defined table of contents exit. This macro is called just before **TC** prints the word 'CONTENTS'. See **TC**.

TY User-defined table of contents exit. This macro is called instead of printing 'CONTENTS'. See **TC**.

VERBON [*flag* [*point-size* [*font*]]]

Begin verbatim output using Courier font. Usually for printing programs. All characters have equal width. The point size can be changed with the second argument. By specifying a third argument it is possible to use another font instead of Courier. *flag* controls several special features. Its value is the sum of all wanted features.

Arg	Description
-----	-------------

- | | |
|----|---|
| 1 | Disable the escape character (\). This is normally turned on during verbose output. |
| 2 | Add an empty line before the verbose text. |
| 4 | Add an empty line after the verbose text. |
| 8 | Print the verbose text with numbered lines. This adds four digit-sized spaces in the beginning of each line. Finer control is available with the string variable Verbnm . It contains all arguments to the troff (1) command .nm , normally '1'. |
| 16 | Indent the verbose text by '5n'. This is controlled by the number-variable Verbin (in units). |

VERBOFF

End verbatim output.

VL *text-indent* [*mark-indent* [**1**]]

Variable-item list. It has no fixed mark, it assumes that every **LI** has a mark instead. *text-indent* sets the indent to the text, and *mark-indent* the distance from the current indentation to the mark. A third argument prohibits printing of a blank line before each item.

VM [**-T**] [*top* [*bottom*]]

Vertical margin. Increase the top and bottom margin by *top* and *bottom*, respectively. If option **-T** is specified, set those margins to *top* and *bottom*. If no argument is given, reset the margin to zero, or to the default ('7v 5v') if **-T** is used. It is highly recommended that macros **TP** and/or **EOP** are defined if using **-T** and setting top and/or bottom margin to less than the default.

WA [*writer-name* [*title*]]

Begin specification of the writer and writer's address. Several names can be specified with empty **WA/WE** pairs, but only one address.

WE End the address specification after **.WA**.

WC [*format1*] [*format2*] [. . .]

Footnote and display width control.

- N** Set default mode which is equal to using the options **-WF**, **-FF**, **-WD**, and **FB**.
- WF** Wide footnotes, wide also in two-column mode.
- WF** Normal footnote width, follow column mode.
- FF** All footnotes gets the same width as the first footnote encountered.
- FF** Normal footnotes, width follows **WF** and **-WF**.
- WD** Wide displays, wide also in two-column mode.
- WD** Normal display width, follow column mode.
- FB** Floating displays generates a line break when printed on the current page.
- FB** Floating displays does not generate line break.

Strings used in mm

App A string containing the word 'APPENDIX'.

Apptxt

The current appendix text.

EM Em dash string

H1txt

Updated by **.H** and **.HU** to the current heading text. Also updated in table of contents & friends.

HF Font list for headings, '2 2 2 2 2 2 2' by default. Non-numeric font names may also be used.

HP Point size list for headings. By default, this is '0 0 0 0 0 0 0' which is the same as '10 10 10 10 10 10 10'.

Index

Contains the string 'INDEX'.

Indcmd

Contains the index command. Default value is 'sort -t\''.

Lifg String containing 'Figure'.

Litb String containing 'TABLE'.

Liex String containing 'Exhibit'.

Liec

String containing 'Equation'.

Licon

String containing 'CONTENTS'.

Lf Contains the string 'LIST OF FIGURES'.

Lt Contains the string 'LIST OF TABLES'.

Lx Contains the string 'LIST OF EXHIBITS'.

Le Contains the string 'LIST OF EQUATIONS'.

Letfc

Contains the string 'Yours very truly', used in **.FC**.

Letapp

Contains the string 'APPROVED:', used in **.AV**.

Letdate

Contains the string 'Date', used in **.AV**.

LetCN

Contains the string 'CONFIDENTIAL', used in **.LO CN**.

LetSA

Contains the string 'To Whom It May Concern:', used in **.LO SA**.

LetAT

Contains the string 'ATTENTION:', used in **.LO AT**.

LetSJ

Contains the string 'SUBJECT:', used in **.LO SJ**.

LetRN

Contains the string 'In reference to:', used in **.LO RN**.

Letns

is an array containing the different strings used in **.NS**. It is really a number of string variables prefixed with **Letns!**. If the argument doesn't exist, it is included between **()** with **Letns!copy** as a prefix and **Letns!to** as a suffix. Observe the space after 'Copy' and before 'to'.

Name	Value
Letns!0	Copy to
Letns!1	Copy (with att.) to
Letns!2	Copy (without att.) to
Letns!3	Att.
Letns!4	Atts.
Letns!5	Enc.
Letns!6	Encs.
Letns!7	Under separate cover
Letns!8	Letter to
Letns!9	Memorandum to
Letns!10	Copy (with atts.) to
Letns!11	Copy (without atts.) to
Letns!12	Abstract Only to
Letns!13	Complete Memorandum to
Letns!14	CC
Letns!copy	Copy (<i>with trailing space</i>)
Letns!to	to(<i>note leading space</i>)

Letnsdef

Define the standard notation used when no argument is given to **.NS**. Default is 0.

MO1 – MO12

Strings containing the month names 'January' through 'December'.

Qrf String containing 'See chapter *[Qrfh], page \n[Qrfp].'

Rp Contains the string 'REFERENCES'.

Tcst

Contains the current status of the table of contents and list of figures, etc. Empty outside of **.TC**. Useful in user-defined macros like **.TP**.

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 Contains the string '\(tm', the trade mark symbol.

Verbnm

Argument to **.nm** in the **.VERBON** command. Default is 1.

Number variables used in mm

Aph Print an appendix page for every new appendix if this number variable is non-zero. No output occurs if **Aph** is zero, but there is always an appendix entry in the 'List of contents'.

Cl Contents level (in the range 0 to 14). The contents is saved if a heading level is lower than or equal to the value of **Cl**. Default is 2.

Cp Eject page between list of table, list of figure, etc., if the value of **Cp** is zero. Default is 0.

D Debug flag. Values greater than zero produce debug information of increasing verbosity. A value of 1 gives information about the progress of formatting. Default is 0.

De If set to 1, eject after floating display is output. Default is 0.

Dsp If defined, it controls the space output before and after static displays. Otherwise the value of **Lsp** is used.

Df Control floating keep output. This is a number in the range 0 to 5, with a default value of 5. See **.DF**.

Ds If set to 1, use the amount of space stored in register **Lsp** before and after display. Default is 1.

Ej If set to 1, eject page before each first-level heading. Default is 0.

Eq Equation labels are left-adjusted if set to 0 and right-adjusted if set to 1. Default is 0.

Fs Footnote spacing. Default is 1.

H1 – H7

Heading counters

H1dot

Append a dot after the level-one heading number if value is greater than zero. Default is 1.

H1h A copy of number register **H1**, but it is incremented just before the page break. Useful in user-defined header macros.

Hb Heading break level. A number in the range 0 to 14, with a default value of 2. See **.H**.

Hc Heading centering level. A number in the range 0 to 14, with a default value value of 0. See **.H**.

Hi Heading temporary indent. A number in the range 0 to 2, with a default value of 1.

- 0 no indentation, left margin
- 1 indent to the right, similar to '**.P 1**'
- 2 indent to line up with text part of preceding heading

Hps Heading pre-space level. If the heading level is less than or equal to **Hps**, two lines precede the section heading instead of one. Default is first level only. The real amount of lines is controlled by the variables **Hps1** and **Hps2**.

Hps1

Number of lines preceding **.H** if the heading level is greater than **Hps**. Value is in units, default is 0.5.

Hps2

Number of lines preceding **.H** if the heading level is less than or equal to **Hps**. Value is in units, default is 1.

Hs Heading space level. A number in the range 0 to 14, with a default value of 2. See **.H**.

Hss Number of lines following **.H** if the heading level is less than or equal to **Hs**. Value is in units, default is 1.

Ht Heading numbering type.

- 0 multiple levels (1.1.1, 1.1.2, etc.)
- 1 single level

Default is 0.

Hu Unnumbered heading level. Default is 2.

Hy Hyphenation status of text body.

- 0 no hyphenation
- 1 hyphenation on, set to value 6

Default is 0.

Iso Set this variable to 1 on the command line to get an ISO-formatted date string (**-rIso=1**). Useless inside of a document.

L Page length, only for command-line settings.

Letwam

Maximum lines in return-address, used in **.WA/WE**. Default is 14.

Lf, Lt, Lx, Le

Enable (1) or disable (0) the printing of List of figures, List of tables, List of

exhibits and List of equations, respectively. Default values are Lf=1, Lt=1, Lx=1, and Le=0.

Li List indentation, used by **.AL**. Default is 6.

Limsp

A flag controlling the insertion of space between prefix and mark in automatic lists (**.AL**).

- 0 no space
- 1 emit space

Ls List space threshold. If current list level is greater than **Ls** no spacing occurs around lists. Default is 99.

Lsp The vertical space used by an empty line. The default is 0.5v in troff mode and 1v in nroff mode.

N Page numbering style.

- 0 normal header for all pages.
- 1 header replaces footer on first page, header is empty.
- 2 page header is removed on the first page.
- 3 'section-page' numbering style enabled.
- 4 page header is removed on the first page.
- 5 'section-page' and 'section-figure' numbering style enabled.

Default is 0. See also the number registers **Sectf** and **Sectp**.

Np A flag to control whether paragraphs are numbered.

- 0 not numbered
- 1 numbered in first-level headings.

Default is 0.

O Page offset, only for command-line settings.

Of Format of figure, table, exhibit, and equation titles.

- 0 ". "
- 1 " - "

Default is 0.

P Current page-number, normally the same as '%' unless 'section-page' numbering style is enabled.

Pi Paragraph indentation. Default is 5.

Pgps

A flag to control whether header and footer point size should follow the current settings or just change when the header and footer are defined.

- 0 Point size only changes to the current setting when **.PH**, **.PF**, **.OH**, **.EH**, **.OF**, or **.OE** is executed.

- 1 Point size changes after every **.S**. This is the default.

Ps Paragraph spacing. Default is 1.

Pt Paragraph type.

- 0 left-justified
- 1 indented paragraphs
- 2 indented paragraphs except after **.H**, **.DE**, or **.LE**.

Default is 0.

Rpe Set default value for second argument of **.RP**. Default is 0.

Sectf

A flag controlling 'section-figures' numbering style. A non-zero value enables this. See also register**N**.

Sectp

A flag controlling 'section-page' numbering style. A non-zero value enables this. See also register**N**.

Si Display indentation. Default is 5.

Verbin

Indentation for **.VERBON**. Default is 5n.

W Line length, only for command-line settings.

.mgm

Always 1.

INTERNALS

The letter macros are using different submacros depending on the letter type. The name of the submacro has the letter type as suffix. It is therefore possible to define other letter types, either in the national macro-file, or as local additions. **.LT** sets the number variables **Pt** and **Pi** to 0 and 5, respectively. The following strings and macros must be defined for a new letter type.

let@init_type

This macro is called directly by **.LT**. It is supposed to initialize variables and other stuff.

let@head_type

This macro prints the letter head, and is called instead of the normal page header. It is supposed to remove the alias **let@header**, otherwise it is called for all pages.

let@sg_type name title n flag [arg1 [arg2 [. . .]]]

.SG is calling this macro only for letters; memorandums have its own processing. *name* and *title* are specified through **.WA/WB**. *n* is the counter, 1-max, and *flag* is true for the last name. Any other argument to **.SG** is appended.

let@fc_type closing

This macro is called by **.FC**, and has the formal closing as the argument.

.LO is implemented as a general option-macro. It demands that a string named **Let-type** is defined, where *type* is the letter type. **.LO** then assigns the argument to the

string variable **let*lo-type**.

FILES

/usr/share/groff/1.22.3.rc1.24-ea225/tmac/m.tmac

/usr/share/groff/1.22.3.rc1.24-ea225/tmac/mm/*.cov

/usr/share/groff/1.22.3.rc1.24-ea225/tmac/mm/*.MT

/usr/share/groff/1.22.3.rc1.24-ea225/tmac/mm/locale

AUTHORS

The GNU version of the *mm* macro package was written by [Jörgen Hägg](#) of Lund, Sweden.

SEE ALSO

groff(1), troff(1), tbl(1), pic(1), eqn(1)

groff_mmse(7)

4.6. mom

See the *groff_mom(7)* man page (type `man groff_mom` at the command line), which gives a short overview and a link to its extensive documentation in HTML format.

NAME

`groff_mom` – groff “mom” macros; “mom” is a “roff” language, part of “groff”

SYNOPSIS

pdfmom [**-Tps**[pdfroff options]] [groff options] *files* ...

groff [**-mom**] *files* ...

groff [**-m mom**] *files* ...

CALLING MOM

mom is a macro set for **groff**, designed primarily to format documents for *PDF* and *PostScript* output.

mom provides two categories of macros: macros for typesetting, and macros for document processing. The typesetting macros provide access to groff’s typesetting capabilities in ways that are simpler to master than groff’s primitives. The document processing macros 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) with or without the `-Tps` option, produce *PDF* documents. The documents include a *PDF* outline that appears in the ‘Contents’ panel of document viewers, and may contain clickable internal and external links.

When `-Tps` is absent, **groff**’s native *PDF* driver, **gropdf**, is used to generate the output. When given, the output is still *PDF*, but processing is passed over to **pdfroff**, which uses **groff**’s PostScript driver, **grops**. Not all *PDF* features are available when `-Tps` is given; its primary use is to allow processing of files with embedded *PostScript* images.

Files processed with **groff -mom** (or `-m mom`) produce *PostScript* output by default.

mom comes with her own very complete documentation in *HTML* format. A separate *PDF manual*, *Producing PDFs with groff and mom*, covers full **mom** or *PDF* usage.

FILES

om.tmac

– the main macro file **mom.tmac** – a wrapper file that calls `om.tmac` directly.

`/usr/share/doc/groff-1.22.3.rc1.24-ea225/html/mom/toc.html`

– entry point to the *HTML* documentation

`/usr/share/doc/groff-1.22.3.rc1.24-ea225/pdf/mom-pdf.pdf`

– the *PDF* manual, *Producing PDFs with groff and mom*

`/usr/share/doc/groff-1.22.3.rc1.24-ea225/examples/mom/*.mom`

– example files using **mom**

DOCUMENTATION IN ALPHABETICAL ORDER

This part of the man page contains information just as in `groff(7)`, *mom macros* and *mom escape sequences* in alphabetical order.

The logical order of *mom macros* and *mom escape sequences* is very well documented in

`/usr/share/doc/groff-1.22.3.rc1.24-ea225/html/mom/toc.html`

– entry point to the HTML documentation

That document is quite good for beginners, but other users should be happy to have some documentation in reference style.

So we restrict this part to the alphabetical order of macros and escape sequences. But, so far, we took all documentation details from the *toc.html* file, just in a more useful alphabetical order. So this part of the man page is nothing new, but only a logical arrangement.

QUICK REFERENCE

Quick Reference of Inline Escape Sequences in alphabetical Order

begin using an initialized colour inline

move backwards in a line

`*[BOLDER]`

invoke pseudo bold inline (related to macro **`.SETBOLDER`**)

`*[BOLDERX]`

off pseudo bold inline (related to macro **`.SETBOLDER`**)

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`**)

pseudo-condensed superscript

temporarily move downwards in a line

`*[EN-MARK]`

mark initial line of a range of line numbers (for use with line numbered end-notes)

`*[EXT]`

invoke pseudo extending inline (related to macro **`.EXTEND`**)

`*[EXTX]`

off pseudo condensing inline (related to macro **`.EXTEND`**)

pseudo extended superscript

move characters pairs further apart inline (related to macro **`.KERN`**)

move forward in a line

`*[LEADER]`

insert leaders at the end of a line

- *[RULE]**
 - draw a full measure rule
 - 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**)
 - string tabs (mark tab positions inline)
 - superscript
- *[TB**
 - inline escape for **.TN** (*Tab Next*)
 - invoke underlining inline (fixed width fonts only)
 - temporarily move upwards in a line

Quick Reference of Macros in alphabetical Order

- .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**

begin using an initialized colour inline

move wards in a line

[BOLDER]**[BOLDERX]**

Emboldening on/off

***[BOLDER]** begins emboldening type. ***[BOLDERX]** turns the feature off. Both are inline escapes, therefore they should not appear as separate lines, but rather be embedded in text lines, like this:

Alternatively, if you wanted the whole line emboldened, you should do
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.

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 escapes, therefore they should not appear as separate lines, but rather be embedded in text lines, like this:

***[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. If you wish the new point size to be pseudo-condensed, simply reinvoke ***[COND]** afterwards. 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.

pseudo-condensed superscript

temporarily move downwards 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 escapes, therefore they should not appear as separate lines, but rather be embedded in text lines, like this:

***[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. If you wish the new point size to be *pseudo-extended*, simply reinvoke ***[EXT]** afterwards. 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.

pseudo extended superscript

move characters pairs further apart inline (related to macro **.KERN**)

move forward in a line

***[LEADER]**

insert leaders at the end of a line

***[RULE]**

draw a full measure rule

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 escapes*, therefore they should not appear as separate lines, but rather be embedded in text lines, like this:

Alternatively, if you wanted the whole line *pseudo-italicized*, you'd do

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**.

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 escapes 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.

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 escapes* (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
\*[RULE] \" Note that you can't use \*[UP] or \*[DOWN] with \*
.RLD 3p
.TQ
```

superscript

***[TB**

Inline escape for **.TN** (*Tab Next*)

invoke underlining inline (fixed width fonts only)

temporarily move upwards 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_FONTWARN

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_FONTTRWARN

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_FONTTRABORT

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 **.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 version of *groff* lower than 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.

If you are running a version of *groff* greater than or equal to 1.19.2, when you invoke the **.FAMILY** macro, **mom** remembers the font style (Roman, **Italic**, etc) currently in use (if the font style exists in the new *family*) and will continue to use the same font style in the new family. For example:

```
.FAMILY BM \Bookmanfamily"
```

```
.FT I \MediumItalic"
<some text> \ " Bookman Medium Italic
.FAMILY H \Helveticafamily"
<more text> \ " Helvetica Medium Italic
```

However, if the font style does not exist in the new family, **mom** will set all subsequent type in the fallback font (by default, **Courier Medium Roman**) until she encounters a **.FT** request 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 \Universfamily"
.FT CD \MediumCondensed"
<some text> \ " Univers Medium Condensed
.FAMILY H \Helveticafamily"
<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 NO TE* 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 greater than or equal to 1.19.2, **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). If your *groff version* is less than 1.19.2, **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. **.I**

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'.

```
.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 afterwards 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. *Amin us 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 afterwards), **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*


```
.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** afterwards, **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.

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 `*[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 *1 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 *NO TE* 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. 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

mom was written by [Peter Schaffter](#) and revised by [Werner Lemberg](#) PDF support was provided by [Deri James](#) The alphabetical documentation of macros and escape sequences in this man page were written by the *mom* team.

SEE ALSO

groff(1), **groff_mom(7)**,

/usr/share/doc/groff-1.22.3.rc1.24-ea225/html/mom/toc.html

– entry point to the HTML documentation

<http://www.schaffter.ca/mom/momdoc/toc.html> – HTML documentation online

<http://www.schaffter.ca/mom/> – the mom macros homepage

BUGS

Please send bug reports to the [groff-bug mailing list](#) or directly to the authors.

5. `gtroff` Reference

This chapter covers **all** of the facilities of `gtroff`. Users of macro packages may skip it if not interested in details.

5.1. Text

`gtroff` input files contain text with control commands interspersed throughout. But, even without control codes, `gtroff` still does several things with the input text:

- filling and adjusting
- adding additional space after sentences
- hyphenating
- inserting implicit line breaks

5.1.1. Filling and Adjusting

When `gtroff` reads text, it collects words from the input and fits as many of them together on one output line as it can. This is known as *filling*.

Once `gtroff` has a *filled* line, it tries to *adjust* it. This means it widens the spacing between words until the text reaches the right margin (in the default adjustment mode). Extra spaces between words are preserved, but spaces at the end of lines are ignored. Spaces at the front of a line cause a *break* (breaks are explained in [Implicit Line Breaks](#)).

See [Manipulating Filling and Adjusting](#).

5.1.2. Hyphenation

Since the odds are not great for finding a set of words, for every output line, which fit nicely on a line without inserting excessive amounts of space between words, `gtroff` hyphenates words so that it can justify lines without inserting too much space between words. It uses an internal hyphenation algorithm (a simplified version of the algorithm used within `TEX`) to indicate which words can be hyphenated and how to do so. When a word is hyphenated, the first part of the word is added to the current filled line being output (with an attached hyphen), and the other portion is added to the next line to be filled.

See [Manipulating Hyphenation](#).

5.1.3. Sentences

Although it is often debated, some typesetting rules say there should be different amounts of space after various punctuation marks. For example, the *Chicago typesetting manual* says that a period at the end of a sentence should have twice as much space following it as would a comma or a period as part of an abbreviation.

`gtroff` does this by flagging certain characters (normally '!', '?', and '.') as *end-of-sentence* characters. When `gtroff` encounters one of these characters at the end of a line, it appends a normal space followed by a *sentence space* in the formatted output. (This justifies one of the conventions mentioned in [Input Conventions](#).)

In addition, the following characters and symbols are treated transparently while handling end-of-sentence characters: `"`, `'`, `)`, `]`, `*`, `\[dg]`, `\[rq]`, and `\[cq]`.

See the `cflags` request in [Using Symbols](#), for more details.

To prevent the insertion of extra space after an end-of-sentence character (at the end of a line), append `\&`.

5.1.4. Tab Stops

`gtroff` translates *tabulator characters*, also called *tabs* (normally code point ASCII `0x09` or EBCDIC `0x05`), in the input into movements to the next tabulator stop. These tab stops are initially located every half inch across the page. Using this, simple tables can be made easily. However, it can often be deceptive as the appearance (and width) of the text on a terminal and the results from `gtroff` can vary greatly.

Also, a possible sticking point is that lines beginning with tab characters are still filled, again producing unexpected results. For example, the following input

```

1      2      3
      4      5

```

produces

```

1      2      3      4      5

```

See [Tabs and Fields](#).

5.1.5. Implicit Line Breaks

An important concept in `gtroff` is the *break*. When a break occurs, `gtroff` outputs the partially filled line (unjustified), and resumes collecting and filling text on the next output line.

There are several ways to cause a break in `gtroff`. A blank line not only causes a break, but it also outputs a one-line vertical space (effectively a blank line). Note that this behaviour can be modified with the blank line macro request `blm`. See [Blank Line Traps](#).

A line that begins with a space causes a break and the space is output at the beginning of the next line. Note that this space isn't adjusted, even in fill mode; however, the behaviour can be modified with the leading spaces macro request `lsm`. See [Leading Spaces Traps](#).

The end of file also causes a break -- otherwise the last line of the document may vanish!

Certain requests also cause breaks, implicitly or explicitly. This is discussed in [Manipulating Filling and Adjusting](#).

5.1.6. Input Conventions

Since `gtroff` does filling automatically, it is traditional in `groff` not to try and type things in as nicely formatted paragraphs. These are some conventions commonly used when typing `gtroff` text:

- Break lines after punctuation, particularly at the end of a sentence and in other logical places. Keep separate phrases on lines by themselves, as entire phrases are often added or deleted when editing.
- Try to keep lines less than 40--60 characters, to allow space for inserting more text.

- Do not try to do any formatting in a WYSIWYG manner (i.e., don't try using spaces to get proper indentation).

5.1.7. Input Encodings

Currently, the following input encodings are available.

`cp1047` This input encoding works only on EBCDIC platforms (and vice versa, the other input encodings don't work with EBCDIC); the file `cp1047.tmac` is by default loaded at start-up.

`latin-1` This is the default input encoding on non-EBCDIC platforms; the file `latin1.tmac` is loaded at start-up.

`latin-2` To use this encoding, either say `'.mso latin2.tmac'` at the very beginning of your document or use `'-mlatin2'` as a command-line argument for `groff`.

`latin-5` For Turkish. Either say `'.mso latin5.tmac'` at the very beginning of your document or use `'-mlatin5'` as a command-line argument for `groff`.

`latin-9 (latin-0)`

This encoding is intended (at least in Europe) to replace `latin-1` encoding. The main difference to `latin-1` is that `latin-9` contains the Euro character. To use this encoding, either say `'.mso latin9.tmac'` at the very beginning of your document or use `'-mlatin9'` as a command-line argument for `groff`.

Note that it can happen that some input encoding characters are not available for a particular output device. For example, saying

```
groff -Tlatin1 -mlatin9 ...
```

fails if you use the Euro character in the input. Usually, this limitation is present only for devices that have a limited set of output glyphs (e.g. `-Tascii` and `-Tlatin1`); for other devices it is usually sufficient to install proper fonts that contain the necessary glyphs.

Due to the importance of the Euro glyph in Europe, the `groff` package now comes with a `POSTSCRIPT` font called `freeeuro.pfa`, which provides various glyph shapes for the Euro. In other words, `latin-9` encoding is supported for the `-Tps` device out of the box (`latin-2` isn't).

By its very nature, `-Tutf8` supports all input encodings; `-Tdvi` has support for both `latin-2` and `latin-9` if the command-line `-mec` is used also to load the file `ec.tmac` (which flips to the EC fonts).

5.2. Measurements

`groff` (like many other programs) requires numeric parameters to specify various measurements. Most numeric parameters⁹ may have a *measurement unit* attached. These units are specified as a single character that immediately follows the number or expression. Each of these units are understood, by `groff`, to be a multiple of its *basic unit*. So, whenever a different measurement unit is specified `groff` converts this into its *basic units*. This basic unit, represented by a 'u', is a device dependent measurement, which is quite small, ranging from 1/75th to 1/72000th of an inch. The values may be given as fractional numbers; however, fractional basic units are always rounded to integers.

⁹ those that specify vertical or horizontal motion or a type size

Some of the measurement units are completely independent of any of the current settings (e.g. type size) of `gtrouff`.

- i Inches. An antiquated measurement unit still in use in certain backwards countries with incredibly low-cost computer equipment. One inch is defined to be 2.54 cm (worldwide since 1964).
- c Centimeters. One centimeter is about 0.3937 in.
- p Points. This is a typesetter's measurement used for measure type size. It is 72 points to an inch.
- P Pica. Another typesetting measurement. 6 picas to an inch (and 12 points to a pica).
- s
- z See [Fractional Type Sizes](#), for a discussion of these units.
- f Fractions. Value is 65536. See [Colors](#), for usage.

The other measurements understood by `gtrouff` depend on settings currently in effect in `gtrouff`. These are very useful for specifying measurements that should look proper with any size of text.

- m Ems. This unit is equal to the current font size in points. So called because it is *approximately* the width of the letter 'm' in the current font.
- n Ens. In `gtrouff`, this is half of an em.
- v Vertical space. This is equivalent to the current line spacing. See [Sizes](#), for more information about this.
- M 100ths of an em.

5.2.1. Default Units

Many requests take a default unit. While this can be helpful at times, it can cause strange errors in some expressions. For example, the line length request expects em units. Here are several attempts to get a line length of 3.5 inches and their results:

```

3.5i      => 3.5i
7/2       => 0i
7/2i     => 0i
(7 / 2)u  => 0i
7i/2     => 0.1i
7i/2u    => 3.5i

```

Everything is converted to basic units first. In the above example it is assumed that 1 i equals 240 u, and 1 m equals 10 p (thus 1 m equals 33 u). The value 7 i/2 is first handled as 7 i/2 m, then converted to 1680 u/66 u, which is 25 u, and this is approximately 0.1 i. As can be seen, a scaling indicator after a closing parenthesis is simply ignored.

Thus, the safest way to specify measurements is to always attach a scaling indicator. If you want to multiply or divide by a certain scalar value, use 'u' as the unit for that value.

5.3. Expressions

`gtroff` has most arithmetic operators common to other languages:

- Arithmetic: '+' (addition), '-' (subtraction), '/' (division), '*' (multiplication), '%' (modulo).
`gtroff` only provides integer arithmetic. The internal type used for computing results is 'int', which is usually a 32-bit signed integer.
- Comparison: '<' (less than), '>' (greater than), '<=' (less than or equal), '>=' (greater than or equal), '=' (equal), '==' (the same as '=').
- Logical: '&' (logical and), '|' (logical or).
- Unary operators: '-' (negating, i.e. changing the sign), '+' (just for completeness; does nothing in expressions), '!' (logical not; this works only within `if` and `while` requests).¹⁰ See below for the use of unary operators in motion requests.

The logical not operator, as described above, works only within `if` and `while` requests. Furthermore, it may appear only at the beginning of an expression, and negates the entire expression. Attempting to insert the '!' operator within the expression results in a 'numeric expression expected' warning. This maintains compatibility with old versions of `tgroff`.

Example:

```
.nr X 1
.nr Y 0
.\" This does not work as expected
.if (\n[X])&(!\n[Y]) .nop X only
.
.\" Use this construct instead
.if (\n[X]=1)&(\n[Y]=0) .nop X only
```

- Extrema: '>?' (maximum), '<?' (minimum).

Example:

```
.nr x 5
.nr y 3
.nr z (\n[x] >? \n[y])
```

The register `z` now contains 5.

- Scaling: $(c; e)$. Evaluate e using c as the default scaling indicator. If c is missing, ignore scaling indicators in the evaluation of e .

Parentheses may be used as in any other language. However, in `gtroff` they are necessary to ensure order of evaluation. `gtroff` has no operator precedence; expressions are evaluated left to right. This means that `gtroff` evaluates '3+5*4' as if it were parenthesized like '(3+5)*4', not as '3+(5*4)', as might be expected.

For many requests that cause a motion on the page, the unary operators '+' and '-' work differently if leading an expression. They then indicate a motion relative to the current

¹⁰ Note that, for example, '!(-1)' evaluates to 'true' because `gtroff` treats both negative numbers and zero as 'false'.

position (down or up, respectively).

Similarly, a leading '[' operator indicates an absolute position. For vertical movements, it specifies the distance from the top of the page; for horizontal movements, it gives the distance from the beginning of the *input* line.

requests and escapes: `bp`, `in`, `ll`, `lt`, `nm`, `nr`, `pl`, `pn`, `po`, `ps`, `pvs`, `rt`, `ti`, `\H`, `\R`, and `\s`. Here, leading plus and minus signs indicate increments and decrements.

See [Setting Registers](#), for some examples. `\B'anything'`

Return 1 if *anything* is a valid numeric expression; or 0 if *anything* is empty or not a valid numeric expression.

Due to the way arguments are parsed, spaces are not allowed in expressions, unless the entire expression is surrounded by parentheses.

See [Request and Macro Arguments](#), and [Conditionals and Loops](#).

5.4. Identifiers

Like any other language, `gtroff` has rules for properly formed *identifiers*. In `gtroff`, an identifier can be made up of almost any printable character, with the exception of the following characters:

- Whitespace characters (spaces, tabs, and newlines).
- Backspace (ASCII `0x08` or EBCDIC `0x16`) and character code `0x01`.
- The following input characters are invalid and are ignored if `gtroff` runs on a machine based on ASCII, causing a warning message of type 'input' (see [Debugging](#), for more details): `0x00`, `0x0B`, `0x0D--0x1F`, `0x80--0x9F`.

And here are the invalid input characters if `gtroff` runs on an EBCDIC host: `0x00`, `0x08`, `0x09`, `0x0B`, `0x0D--0x14`, `0x17--0x1F`, `0x30--0x3F`.

Currently, some of these reserved codepoints are used internally, thus making it non-trivial to extend `gtroff` to cover Unicode or other character sets and encodings that use characters of these ranges.

Note that invalid characters are removed before parsing; an identifier `foo`, followed by an invalid character, followed by `bar` is treated as `foobar`.

For example, any of the following is valid.

```
br
PP
(l
end-list
@_
```

Note that identifiers longer than two characters with a closing bracket (']') in its name can't be accessed with escape sequences that expect an identifier as a parameter. For example, `\[foo]]` accesses the glyph 'foo', followed by ']', whereas `\C'foo]]` really asks for glyph 'foo]'].

To avoid problems with the `refer` preprocessor, macro names should not start with '[' or ']'. Due to backwards compatibility, everything after '[' and ']' is handled as a special argument to `refer`. For example, '['foo' makes `refer` to start a reference, using 'foo' as a

parameter.

`\A'ident'`

Test whether an identifier *ident* is valid in `gtrouff`. It expands to the character 1 or 0 according to whether its argument (usually delimited by quotes) is or is not acceptable as the name of a string, macro, diversion, number register, environment, or font. It returns 0 if no argument is given. This is useful for looking up user input in some sort of associative table.

```
\A'end-list'
=> 1
```

See [Escapes](#), for details on parameter delimiting characters. Identifiers in `gtrouff` can be any length, but, in some contexts, `gtrouff` needs to be told where identifiers end and text begins (and in different ways depending on their length):

- Single character.
- Two characters. Must be prefixed with '(' in some situations.
- Arbitrary length (`gtrouff` only). Must be bracketed with '[' and ']' in some situations. Any length identifier can be put in brackets.

Unlike many other programming languages, undefined identifiers are silently ignored or expanded to nothing. When `gtrouff` finds an undefined identifier, it emits a warning, doing the following:

- If the identifier is a string, macro, or diversion, `gtrouff` defines it as empty.
- If the identifier is a number register, `gtrouff` defines it with a value of 0.

See [Warnings.](#), [Interpolating Registers](#), and [Strings](#). Note that macros, strings, and diversions share the same name space.

```
.de xxx
.  nop foo
..
.
.di xxx
bar
.br
.di
.
.xxx
=> bar
```

As can be seen in the previous example, `gtrouff` reuses the identifier 'xxx', changing it from a macro to a diversion. No warning is emitted! The contents of the first macro definition is lost.

See [Interpolating Registers](#), and [Strings](#).

5.5. Embedded Commands

Most documents need more functionality beyond filling, adjusting and implicit line breaking. In order to gain further functionality, `gtrouff` allows commands to be embedded into the

text, in two ways.

The first is a *request* that takes up an entire line, and does some large-scale operation (e.g. break lines, start new pages).

The other is an *escape* that can be usually embedded anywhere in the text; most requests can accept it even as an argument. Escapes generally do more minor operations like sub- and superscripts, print a symbol, etc.

5.5.1. Requests

A request line begins with a control character, which is either a single quote (‘’, the *no-break control character*) or a period (‘.’, the normal *control character*). These can be changed; see [Character Translations](#), for details. After this there may be optional tabs or spaces followed by an identifier, which is the name of the request. This may be followed by any number of space-separated arguments (*no* tabs here).

Since a control character followed by whitespace only is ignored, it is common practice to use this feature for structuring the source code of documents or macro packages.

```
.de foo
.  tm This is foo.
..
.
.
.de bar
.  tm This is bar.
..
```

Another possibility is to use the blank line macro request `blm` by assigning an empty macro to it.

```
.de do-nothing
..
.blm do-nothing  \" activate blank line macro

.de foo
.  tm This is foo.
..

.de bar
.  tm This is bar.
..

.blm                \" deactivate blank line macro
```

See [Blank Line Traps](#). To begin a line with a control character without it being interpreted, precede it with `\&`. This represents a zero width space, which means it does not affect the output.

In most cases the period is used as a control character. Several requests cause a break implicitly; using the single quote control character prevents this.

`\n[.br]`

A read-only number register, which is set to 1 if a macro is called with the normal control character (as defined with the `cc` request), and set to 0 otherwise.

This allows reliable modification of requests.

```
.als bp*orig bp
.de bp
. tm before bp
. ie \n[.br] .bp*orig
. el 'bp*orig
. tm after bp
..
```

Using this register outside of a macro makes no sense (it always returns zero in such cases).

If a macro is called as a string (that is, using `*`), the value of the `.br` register is inherited from the caller.

5.5.1.1. Request and Macro Arguments

Arguments to requests and macros are processed much like the shell: The line is split into arguments according to spaces.¹¹

An argument to a macro that is intended to contain spaces can either be enclosed in double quotes, or have the spaces *escaped* with backslashes. This is *not* true for requests.

Here are a few examples for a hypothetical macro `uh`:

```
.uh The Mouse Problem
.uh "The Mouse Problem"
.uh The\ Mouse\ Problem
```

The first line is the `uh` macro being called with 3 arguments, ‘The’, ‘Mouse’, and ‘Problem’. The latter two have the same effect of calling the `uh` macro with one argument, ‘The Mouse Problem’.¹²

A double quote that isn’t preceded by a space doesn’t start a macro argument. If not closing a string, it is printed literally.

For example,

```
.xxx a" "b c" "de"fg"
```

has the arguments ‘a”, ‘b c’, ‘de’, and ‘fg’’. Don’t rely on this obscure behaviour!

There are two possibilities to get a double quote reliably.

- Enclose the whole argument with double quotes and use two consecutive double quotes to represent a single one. This traditional solution has the disadvantage that double quotes don’t survive argument expansion again if called in compatibility mode (using the `-C` option of `groff`):

¹¹ Plan 9’s `troff` implementation also allows tabs for argument separation -- `gtroff` intentionally doesn’t support this.

¹² The last solution, i.e., using escaped spaces, is “classical” in the sense that it can be found in most `troff` documents. Nevertheless, it is not optimal in all situations, since `\ ’` inserts a fixed-width, non-breaking space character that can’t stretch. `gtroff` provides a different command `\~` to insert a stretchable, non-breaking space.

```

.de xx
.  tm xx:  `\$1'  `\$2'  `\$3'
.
.  yy "\$1" "\$2" "\$3"
..
.de yy
.  tm yy:  `\$1'  `\$2'  `\$3'
..
.xx A "test with ""quotes"" .
    => xx: `A' `test with "quotes"' `.'
    => yy: `A' `test with ' `quotes'"'"'

```

If not in compatibility mode, you get the expected result

```

xx: `A' `test with "quotes"' `.'
yy: `A' `test with "quotes"' `.'

```

since `gtrouff` preserves the input level.

- Use the double quote glyph `\(dq`. This works with and without compatibility mode enabled since `gtrouff` doesn't convert `\(dq` back to a double quote input character.

Note that this method won't work with UNIX `trouff` in general since the glyph `'dq` isn't defined normally.

Double quotes in the `ds` request are handled differently. See [Strings](#), for more details.

5.5.2. Macros

`gtrouff` has a *macro* facility for defining a series of lines that can be invoked by name. They are called in the same manner as requests -- arguments also may be passed basically in the same manner.

See [Writing Macros](#), and [Request and Macro Arguments](#).

5.5.3. Escapes

Escapes may occur anywhere in the input to `gtrouff`. They usually begin with a backslash and are followed by a single character, which indicates the function to be performed. The escape character can be changed; see [Character Translations](#).

Escape sequences that require an identifier as a parameter accept three possible syntax forms.

- The next single character is the identifier.
- If this single character is an opening parenthesis, take the following two characters as the identifier. Note that there is no closing parenthesis after the identifier.
- If this single character is an opening bracket, take all characters until a closing bracket as the identifier.

Examples:

```

\fb
\n(XX

```

```
\*[TeX]
```

Other escapes may require several arguments and/or some special format. In such cases the argument is traditionally enclosed in single quotes (and quotes are always used in this manual for the definitions of escape sequences). The enclosed text is then processed according to what that escape expects. Example:

```
\l'1.5i\ (bu'
```

Note that the quote character can be replaced with any other character that does not occur in the argument (even a newline or a space character) in the following escapes: `\o`, `\b`, and `\x`. This makes e.g.

```
A caf
\b
e\'
```

```
in Paris
=> A café in Paris
```

possible, but it is better not to use this feature to avoid confusion.

The following escapes sequences (which are handled similarly to characters since they don't take a parameter) are also allowed as delimiters: `\%`, `\'`, `\|`, `\^`, `\{`, `\}`, `\'`, `\'`, `\-`, `_`, `\!`, `\?`, `\)`, `\/`, `\,`, `\&`, `\:`, `\~`, `\0`, `\a`, `\c`, `\d`, `\e`, `\E`, `\p`, `\r`, `\t`, and `\u`. Again, don't use these if possible.

No newline characters as delimiters are allowed in the following escapes: `\A`, `\B`, `\Z`, `\C`, and `\w`.

Finally, the escapes `\D`, `\h`, `\H`, `\l`, `\L`, `\N`, `\R`, `\s`, `\S`, `\v`, and `\x` can't use the following characters as delimiters:

- The digits 0-9.
- The (single-character) operators `'+/*%<>=&:()'`.
- The space, tab, and newline characters.
- All escape sequences except `\%`, `\:`, `\{`, `\}`, `\'`, `\'`, `\-`, `_`, `\!`, `\/`, `\c`, `\e`, and `\p`.

To have a backslash (actually, the current escape character) appear in the output several escapes are defined: `\`, `\e` or `\E`. These are very similar, and only differ with respect to being used in macros or diversions. See [Character Translations](#), for an exact description of those escapes.

See [Implementation Differences](#), [Copy-in Mode](#), and

[Diversions](#), [Identifiers](#), for more information.

5.5.3.1. Comments

Probably one of the most¹³ common forms of escapes is the comment.

¹³ Unfortunately, this is a lie. But hopefully future `gtroff` hackers will believe it :-)

`\"`

Start a comment. Everything to the end of the input line is ignored.

This may sound simple, but it can be tricky to keep the comments from interfering with the appearance of the final output.

If the escape is to the right of some text or a request, that portion of the line is ignored, but the space leading up to it is noticed by `gtroff`. This only affects the `ds` and `as` request and its variants.

One possibly irritating idiosyncrasy is that tabs must not be used to line up comments. Tabs are not treated as whitespace between the request and macro arguments.

A comment on a line by itself is treated as a blank line, because after eliminating the comment, that is all that remains:

```
Test
\" comment
Test
```

produces

```
Test
```

```
Test
```

To avoid this, it is common to start the line with `.\"`, which causes the line to be treated as an undefined request and thus ignored completely.

Another commenting scheme seen sometimes is three consecutive single quotes (`'''`) at the beginning of a line. This works, but `gtroff` gives a warning about an undefined macro (namely `''`), which is harmless, but irritating.

`\#`

To avoid all this, `gtroff` has a new comment mechanism using the `\#` escape. This escape works the same as `\"` except that the newline is also ignored:

```
Test
\# comment
Test
```

produces

```
Test Test
```

as expected.

`.ig [end]`

Ignore all input until `gtroff` encounters the macro named `.end` on a line by itself (or `..` if `end` is not specified). This is useful for commenting out large blocks of text:

```
text text text...
.ig
This is part of a large block
of text that has been
temporarily(?) commented out.
```

We can restore it simply by removing the `.ig` request and the `..` at the

```

end of the block.
..
More text text text...

```

produces

```

text text text... More text text text...

```

Note that the commented-out block of text does not cause a break.

The input is read in copy-mode; auto-incremented registers *are* affected (see [Auto-increment](#)).

5.6. Registers

Numeric variables in `gtroff` are called *registers*. There are a number of built-in registers, supplying anything from the date to details of formatting parameters.

See [Identifiers](#), for details on register identifiers.

5.6.1. Setting Registers

Define or set registers using the `nr` request or the `\R` escape.

Although the following requests and escapes can be used to create registers, simply using an undefined register will cause it to be set to zero.

```

.nr ident value
\R'ident value'

```

Set number register *ident* to *value*. If *ident* doesn't exist, `gtroff` creates it.

The argument to `\R` usually has to be enclosed in quotes. See [Escapes](#), for details on parameter delimiting characters.

The `\R` escape doesn't produce an input token in `gtroff`; in other words, it vanishes completely after `gtroff` has processed it.

For example, the following two lines are equivalent:

```

.nr a (((17 + (3 * 4))) % 4)
\R'a (((17 + (3 * 4))) % 4)'
=> 1

```

Note that the complete transparency of `\R` can cause surprising effects if you use number registers like `.k`, which get evaluated at the time they are accessed.

```

.ll 1.6i
.
aaa bbb ccc ddd eee fff ggg hhh\R':k \n[.k]'
.tm :k == \n[:k]
=> :k == 126950
.
.br
.
aaa bbb ccc ddd eee fff ggg hhh\h'0'\R':k \n[.k]'
.tm :k == \n[:k]
=> :k == 15000

```

If you process this with the `POSTSCRIPT` device (`-Tps`), there will be a line break

eventually after `ggg` in both input lines. However, after processing the space after `ggg`, the partially collected line is not overfull yet, so `t_rouff` continues to collect input until it sees the space (or in this case, the newline) after `hhh`. At this point, the line is longer than the line length, and the line gets broken.

In the first input line, since the `\R` escape leaves no traces, the check for the overfull line hasn't been done yet at the point where `\R` gets handled, and you get a value for the `.k` number register that is even greater than the current line length.

In the second input line, the insertion of `\h'0'` to emit an invisible zero-width space forces `t_rouff` to check the line length, which in turn causes the start of a new output line. Now `.k` returns the expected value.

Both `nr` and `\R` have two additional special forms to increment or decrement a register.

```
.nr ident +value
```

```
.nr ident -value
```

```
\R'ident +value'
```

```
\R'ident -value'
```

Increment (decrement) register *ident* by *value*.

```
.nr a 1
.nr a +1
\na
=> 2
```

To assign the negated value of a register to another register, some care must be taken to get the desired result:

```
.nr a 7
.nr b 3
.nr a -\nb
\na
=> 4
.nr a (-\nb)
\na
=> -3
```

The surrounding parentheses prevent the interpretation of the minus sign as a decrementing operator. An alternative is to start the assignment with a '0':

```
.nr a 7
.nr b -3
.nr a \nb
\na
=> 4
.nr a 0\nb
\na
=> -3
```

```
.rr ident
```

Remove number register *ident*. If *ident* doesn't exist, the request is ignored.

```
.rnn ident1 ident2
```

Rename number register *ident1* to *ident2*. If either *ident1* or *ident2* doesn't exist, the request is ignored.

```
.aln ident1 ident2
```

Create an alias *ident1* for a number register *ident2*. The new name and the old name are exactly equivalent. If *ident1* is undefined, a warning of type 'reg' is generated, and the request is ignored. See [Debugging](#), for information about warnings.

5.6.2. Interpolating Registers

Numeric registers can be accessed via the `\n` escape.

```
\ni
\n(id
\n[ident]
```

Interpolate number register with name *ident* (one-character name *i*, two-character name *id*). This means that the value of the register is expanded in-place while `gtroff` is parsing the input line. Nested assignments (also called indirect assignments) are possible.

```
.nr a 5
.nr as \na+\na
\n(as
=> 10

.nr a1 5
.nr ab 6
.ds str b
.ds num 1
\n[a\n[num]]
=> 5
\n[a\*[str]]
=> 6
```

5.6.3. Auto-increment

Number registers can also be auto-incremented and auto-decremented. The increment or decrement value can be specified with a third argument to the `nr` request or `\R` escape.

```
.nr ident value incr
```

Set number register *ident* to *value*; the increment for auto-incrementing is set to *incr*. Note that the `\R` escape doesn't support this notation.

To activate auto-incrementing, the escape `\n` has a special syntax form.

```
\n+i
\n-i
\n+(id
\n-(id
\n+[ident]
\n-[ident]
```

Before interpolating, increment or decrement *ident* (one-character name *i*, two-character name *id*) by the auto-increment value as specified with the `nr` request (or the `\R` escape). If no auto-increment value has been specified, these syntax forms are identical to `\n`.

For example,

```
.nr a 0 1
.nr xx 0 5
.nr foo 0 -2
\n+a, \n+a, \n+a, \n+a, \n+a
.br
\n-(xx, \n-(xx, \n-(xx, \n-(xx, \n-(xx
.br
\n+[foo], \n+[foo], \n+[foo], \n+[foo], \n+[foo]
```

produces

```
1, 2, 3, 4, 5
-5, -10, -15, -20, -25
-2, -4, -6, -8, -10
```

To change the increment value without changing the value of a register (*a* in the example), the following can be used:

```
.nr a \na 10
```

5.6.4. Assigning Formats

When a register is used, it is always textually replaced (or interpolated) with a representation of that number. This output format can be changed to a variety of formats (numbers, Roman numerals, etc.). This is done using the `af` request.

`.af ident format`

Change the output format of a number register. The first argument *ident* is the name of the number register to be changed, and the second argument *format* is the output format. The following output formats are available:

- | | |
|-------|--|
| 1 | Decimal arabic numbers. This is the default format: 0, 1, 2, 3, ... |
| 0...0 | Decimal numbers with as many digits as specified. So, '00' would result in printing numbers as 01, 02, 03, ...

In fact, any digit instead of zero does work; <code>gtroff</code> only counts how many digits are specified. As a consequence, <code>af</code> 's default format '1' could be specified as '0' also (and exactly this is returned by the <code>\g</code> escape, see below). |
| I | Upper-case Roman numerals: 0, I, II, III, IV, ... |
| i | Lower-case Roman numerals: 0, i, ii, iii, iv, ... |
| A | Upper-case letters: 0, A, B, C, ..., Z, AA, AB, ... |
| a | Lower-case letters: 0, a, b, c, ..., z, aa, ab, ... |

Omitting the number register format causes a warning of type 'missing'. See [Debugging](#), for more details. Specifying a nonexistent format causes an error.

The following example produces '10, X, j, 010':

```
.nr a 10
.af a 1          \" the default format
```

```

\na,
.af a I
\na,
.af a a
\na,
.af a 001
\na

```

The largest number representable for the ‘i’ and ‘l’ formats is 39999 (or –39999); UNIX `trouff` uses ‘z’ and ‘w’ to represent 10000 and 5000 in Roman numerals, and so does `gtrouff`. Currently, the correct glyphs of Roman numeral five thousand and Roman numeral ten thousand (Unicode code points U+2182 and U+2181, respectively) are not available.

If *ident* doesn’t exist, it is created.

Changing the output format of a read-only register causes an error. It is necessary to first copy the register’s value to a writeable register, then apply the `af` request to this other register.

```

\gi
\g(id
\g[ident]

```

Return the current format of the specified register *ident* (one-character name *i*, two-character name *id*). For example, ‘ga’ after the previous example would produce the string ‘000’. If the register hasn’t been defined yet, nothing is returned.

5.6.5. Built-in Registers

The following lists some built-in registers that are not described elsewhere in this manual. Any register that begins with a ‘.’ is read-only. A complete listing of all built-in registers can be found in [Register Index](#).

- `\n[.F]` This string-valued register returns the current input file name.
- `\n[.H]` Horizontal resolution in basic units.
- `\n[.R]` The number of number registers available. This is always 10000 in GNU `trouff`; it exists for backward compatibility.
- `\n[.U]` If `gtrouff` is called with the `-U` command-line option to activate unsafe mode, the number register `.U` is set to 1, and to zero otherwise. See [Options](#).
- `\n[.V]` Vertical resolution in basic units.
- `\n[seconds]`
The number of seconds after the minute, normally in the range 0 to 59, but can be up to 61 to allow for leap seconds. Initialized at start-up of `gtrouff`.
- `\n[minutes]`
The number of minutes after the hour, in the range 0 to 59. Initialized at start-up of `gtrouff`.
- `\n[hours]`
The number of hours past midnight, in the range 0 to 23. Initialized at start-up of

gtroff.

- `\n[dw]` Day of the week (1--7).
- `\n[dy]` Day of the month (1--31).
- `\n[mo]` Current month (1--12).
- `\n[year]`
The current year.
- `\n[yr]` The current year minus 1900. Unfortunately, the documentation of UNIX Version 7's `troff` had a year 2000 bug: It incorrectly claimed that `yr` contains the last two digits of the year. That claim has never been true of either AT&T `troff` or GNU `troff`. Old `troff` input that looks like this:
- ```
'\" The following line stopped working after 1999
This document was formatted in 19\n(yr.
```
- can be corrected as follows:
- ```
This document was formatted in \n[year].
```
- or, to be portable to older `troff` versions, as follows:
- ```
.nr y4 1900+\n(yr
This document was formatted in \n(y4.
```
- `\n[.c]`
- `\n[c.]` The current *input* line number. Register `'c'` is read-only, whereas `'c.'` (`agtroff` extension) is writable also, affecting both `'c'` and `'c.'`
- `\n[ln]` The current *output* line number after a call to the `nm` request to activate line numbering.  
See [Miscellaneous](#), for more information about line numbering.
- `\n[.x]` The major version number. For example, if the version number is 1.03 then `.x` contains `'1'`.
- `\n[.y]` The minor version number. For example, if the version number is 1.03 then `.y` contains `'03'`.
- `\n[.Y]` The revision number of `groff`.
- `\n[$$]` The process ID of `gtroff`.
- `\n[.g]` Always 1. Macros should use this to determine whether they are running under GNU `troff`.
- `\n[.A]` If the command-line option `-a` is used to produce an ASCII approximation of the output, this is set to 1, zero otherwise. See [Options](#).
- `\n[.O]` This read-only register is set to the suppression nesting level (see escapes `\O`). See [Suppressing output](#).
- `\n[.P]` This register is set to 1 (and to 0 otherwise) if the current page is actually being printed, i.e., if the `-o` option is being used to only print selected pages. See [Options](#), for more information.

- `\n[.T]` If `gtroff` is called with the `-T` command-line option, the number register `.T` is set to 1, and zero otherwise. See [Options](#).
- `\*[.T]` A single read-write string register that contains the current output device (for example, 'latin1' or 'ps'). This is the only string register defined by `gtroff`.

## 5.7. Manipulating Filling and Adjusting

Various ways of causing *breaks* were given in [Implicit Line Breaks](#). The `br` request likewise causes a break. Several other requests also cause breaks, but implicitly. These are `bp`, `ce`, `cf`, `fi`, `fl`, `in`, `nf`, `rj`, `sp`, `ti`, and `trf`.

`.br`

Break the current line, i.e., the input collected so far is emitted without adjustment.

If the no-break control character is used, `gtroff` suppresses the break:

```

a
'br
b
=> a b

```

Initially, `gtroff` fills and adjusts text to both margins. Filling can be disabled via the `nf` request and re-enabled with the `fi` request.

`.fi`

`\n[.u]`

Activate fill mode (which is the default). This request implicitly enables adjusting; it also inserts a break in the text currently being filled. The read-only number register `.u` is set to 1.

The fill mode status is associated with the current environment (see [Environments](#)).

See [Line Control](#), for interaction with the `\c` escape.

`.nf`

Activate no-fill mode. Input lines are output as-is, retaining line breaks and ignoring the current line length. This command implicitly disables adjusting; it also causes a break. The number register `.u` is set to 0.

The fill mode status is associated with the current environment (see [Environments](#)).

See [Line Control](#), for interaction with the `\c` escape.

`.ad [mode]`

`\n[.j]`

Set adjusting mode.

Activation and deactivation of adjusting is done implicitly with calls to the `fi` or `nf` requests.

*mode* can have one of the following values:

- |   |                                                                                               |
|---|-----------------------------------------------------------------------------------------------|
| l | Adjust text to the left margin. This produces what is traditionally called ragged-right text. |
| r | Adjust text to the right margin, producing ragged-left text.                                  |

`c` Center filled text. This is different to the `ce` request, which only centers text without filling.

`b`

`n` Justify to both margins. This is the default used by `gtroff`.

Finally, *mode* can be the numeric argument returned by the `.j` register.

Using `ad` without argument is the same as saying `.ad \[.j]`. In particular, `gtroff` adjusts lines in the same way it did before adjusting was deactivated (with a call to `na`, say). For example, this input code

```
.de AD
. br
. ad \$1
..
.
.de NA
. br
. na
..
.
textA
.AD r
.nr ad \n[.j]
textB
.AD c
textC
.NA
textD
.AD \" back to centering
textE
.AD \n[ad] \" back to right justifying
textF
```

produces the following output:

```
textA
 textB
 textC
textD
 textE
 textF
```

As just demonstrated, the current adjustment mode is available in the read-only number register `.j`; it can be stored and subsequently used to set adjustment.

The adjustment mode status is associated with the current environment (see [Environments](#)).

`.na`

Disable adjusting. This request won't change the current adjustment mode: A subsequent call to `ad` uses the previous adjustment setting.

The adjustment mode status is associated with the current environment (see

[Environments](#)).

```
.brp
\p
```

Adjust the current line and cause a break.

In most cases this produces very ugly results since `gtroff` doesn't have a sophisticated paragraph building algorithm (as `TEX` have, for example); instead, `gtroff` fills and adjusts a paragraph line by line:

```
This is an uninteresting sentence.
This is an uninteresting sentence.\p
This is an uninteresting sentence.
```

is formatted as

```
This is an uninteresting sentence. This is an
uninteresting sentence.
This is an uninteresting sentence.
```

```
.ss word_space_size [sentence_space_size]
\n[.ss]
\n[.sss]
```

Change the size of a space between words. It takes its units as one twelfth of the space width parameter for the current font. Initially both the *word\_space\_size* and *sentence\_space\_size* are 12. In fill mode, the values specify the minimum distance.

If two arguments are given to the `ss` request, the second argument sets the sentence space size. If the second argument is not given, sentence space size is set to *word\_space\_size*. The sentence space size is used in two circumstances: If the end of a sentence occurs at the end of a line in fill mode, then both an inter-word space and a sentence space are added; if two spaces follow the end of a sentence in the middle of a line, then the second space is a sentence space. If a second argument is never given to the `ss` request, the behaviour of UNIX `troff` is the same as that exhibited by GNU `troff`. In GNU `troff`, as in UNIX `troff`, a sentence should always be followed by either a newline or two spaces.

The read-only number registers `.ss` and `.sss` hold the values of the parameters set by the first and second arguments of the `ss` request.

The word space and sentence space values are associated with the current environment (see [Environments](#)).

Contrary to AT&T `troff`, this request is *not* ignored if a TTY output device is used; the given values are then rounded down to a multiple of 12 (see [Implementation Differences](#)).

The request is ignored if there is no parameter.

Another useful application of the `ss` request is to insert discardable horizontal space, i.e., space that is discarded at a line break. For example, paragraph-style footnotes could be separated this way:

```
.ll 4.5i
1.\ This is the first footnote.\c
.ss 48
.nop
.ss 12
```

```
2.\ This is the second footnote.
```

The result:

```
1. This is the first footnote. 2. This
 is the second footnote.
```

Note that the `\h` escape produces unbreakable space.

```
.ce [nnn]
```

```
\n[.ce]
```

Center text. While the `'ad c'` request also centers text, it fills the text as well. `ce` does not fill the text it affects. This request causes a break. The number of lines still to be centered is associated with the current environment (see [Environments](#)).

The following example demonstrates the differences. Here is the input:

```
.ll 4i
.ce 1000
This is a small text fragment that shows the differences
between the '.ce' and the '.ad c' request.
.ce 0

.ad c
This is a small text fragment that shows the differences
between the '.ce' and the '.ad c' request.
```

And here the result:

```
 This is a small text fragment that
 shows the differences
between the '.ce' and the '.ad c' request.

 This is a small text fragment that
shows the differences between the '.ce'
 and the '.ad c' request.
```

With no arguments, `ce` centers the next line of text. `nnn` specifies the number of lines to be centered. If the argument is zero or negative, centering is disabled.

The basic length for centering text is the line length (as set with the `ll` request) minus the indentation (as set with the `in` request). Temporary indentation is ignored.

As can be seen in the previous example, it is a common idiom to turn on centering for a large number of lines, and to turn off centering after text to be centered. This is useful for any request that takes a number of lines as an argument.

The `.ce` read-only number register contains the number of lines remaining to be centered, as set by the `ce` request.

```
.rj [nnn]
```

```
\n[.rj]
```

Justify unfilled text to the right margin. Arguments are identical to the `ce` request. The `.rj` read-only number register is the number of lines to be right-justified as set by the `rj` request. This request causes a break. The number of lines still to be right-justified is associated with the current environment (see [Environments](#)).

## 5.8. Manipulating Hyphenation

Here a description of requests that influence hyphenation.

`.hy [mode]`

`\n[.hy]`

Enable hyphenation. The request has an optional numeric argument, *mode*, to restrict hyphenation if necessary:

- |    |                                                                                                                                                                                                     |
|----|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 1  | The default argument if <i>mode</i> is omitted: Hyphenation is enabled, and the first and the last character of a word is not hyphenated. This is also the start-up value of <code>gtrouff</code> . |
| 2  | Do not hyphenate the last word on a page or column.                                                                                                                                                 |
| 4  | Do not hyphenate the last two characters of a word.                                                                                                                                                 |
| 8  | Do not hyphenate the first two characters of a word.                                                                                                                                                |
| 16 | Allow hyphenation before the last character of a word.                                                                                                                                              |
| 32 | Allow hyphenation after the first character of a word.                                                                                                                                              |

The values in the previous table are additive. For example, value 12 causes `gtrouff` to neither hyphenate the last two nor the first two characters of a word. Note that value 13 would do exactly the same; in other words, value 1 need not be added if the value is larger than 1.

Using values 4 and 16 can't be used together since they contradict each other; the same holds for values 8 and 32.

The number of characters at the beginning of a word after which the first hyphenation point should be inserted is determined by the patterns themselves; it can't be reduced further without introducing additional, invalid hyphenation points (unfortunately, this information is not part of a pattern file, you have to know it in advance). The same is true for the number of characters at the end of word before the last hyphenation point should be inserted. For example, the code

```
.ll 1
.hy 48
```

returns

```
s-
plit-
t-
in-
g
```

instead of the correct 'split-ting'. US-English patterns as distributed with `gtrouff` need two characters at the beginning and three characters at the end; this means that value 4 of `hy` is mandatory. Value 8 is possible as an additional restriction, but values 1 (the default!), 16, and 32 should be avoided.

Here is a table of left and right minimum values for hyphenation as needed by the patterns distributed with `gtrouff`; see the *gtrouff\_tmac(5) man page* (type `man gtrouff_tmac` at the command line) for more information on `gtrouff`'s language

macro files.

| <b>language</b>    | <b>pattern name</b> | <b>left min</b> | <b>right min</b> |
|--------------------|---------------------|-----------------|------------------|
| Czech              | cs                  | 2               | 2                |
| US English         | us                  | 2               | 3                |
| French             | fr                  | 2               | 3                |
| German traditional | det                 | 2               | 2                |
| German reformed    | den                 | 2               | 2                |
| Swedish            | sv                  | 1               | 2                |

Hyphenation exceptions within pattern files (i.e., the words within a `\hyphenation` group) also obey the hyphenation restrictions given by `hy`. However, exceptions specified with the `hw` do not.

The current hyphenation restrictions can be found in the read-only number register `‘.hy’`.

The hyphenation mode is associated with the current environment (see [Environments](#)).

`.nh`

Disable hyphenation (i.e., set the hyphenation mode to zero). Note that the hyphenation mode of the last call to `hy` is not remembered.

The hyphenation mode is associated with the current environment (see [Environments](#)).

`.hlm [nnn]`

`\n[.hlm]`

`\n[.hlc]`

Set the maximum number of consecutive hyphenated lines to *nnn*. If this number is negative, there is no maximum. The default value is `-1` if *nnn* is omitted. This value is associated with the current environment (see [Environments](#)). Only lines output from a given environment count towards the maximum associated with that environment. Hyphens resulting from `\%` are counted; explicit hyphens are not.

The current setting of `hlm` is available in the `.hlm` read-only number register. Also the number of immediately preceding consecutive hyphenated lines are available in the read-only number register `‘.hlc’`.

`.hw word1 word2 ...`

Define how *word1*, *word2*, etc. are to be hyphenated. The words must be given with hyphens at the hyphenation points. For example:

```
.hw in-sa-lub-rious
```

Besides the space character, any character whose hyphenation code value is zero can be used to separate the arguments of `hw` (see the documentation for the `hcode` request below for more information). In addition, this request can be used more than once.

Hyphenation points specified with `hw` are not subject to the restrictions given by the `hy` request.

Hyphenation exceptions specified with the `hw` request are associated with the current hyphenation language; it causes an error if there is no current hyphenation language.

This request is ignored if there is no parameter.

In old versions of `ttrouff` there was a limited amount of space to store such information; fortunately, with `gtrouff`, this is no longer a restriction.

`\%`

To tell `gtrouff` how to hyphenate words on the fly, use the `\%` escape, also known as the *hyphenation character*. Preceding a word with this character prevents it from being hyphenated; putting it inside a word indicates to `gtrouff` that the word may be hyphenated at that point. Note that this mechanism only affects that one occurrence of the word; to change the hyphenation of a word for the entire document, use the `hw` request.

The `\:` escape inserts a zero-width break point (that is, the word breaks but without adding a hyphen).

```
... check the /var/log/\:httpd/\:access_log file ...
```

Note that `\X` and `\Y` start a word, that is, the `\%` escape in (say) `'\X'...\%foobar'` and `'\Y'...\%foobar'` no longer prevents hyphenation but inserts a hyphenation point at the beginning of `'foobar'`; most likely this isn't what you want to do.

`.hc [char]`

Change the hyphenation character to *char*. This character then works the same as the `\%` escape, and thus, no longer appears in the output. Without an argument, `hc` resets the hyphenation character to be `\%` (the default) only.

The hyphenation character is associated with the current environment (see [Environments](#)).

`.hpf pattern_file`

`.hpfa pattern_file`

`.hpfcodes a b [c d ...]`

Read in a file of hyphenation patterns. This file is searched for in the same way as `name.tmac` (or `tmac.name`) is searched for if the `-mname` option is specified.

It should have the same format as (simple) `TEX` patterns files. More specifically, 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.
- No support for 'digraphs' like `\$`.
- `^^xx` (*x* is 0--9 or a--f) and `^^x` (character code of *x* in the range 0--127) are recognized; other use of `^` causes an error.
- No macro expansion.
- `hpf` checks for the expression `\patterns{...}` (possibly with whitespace before and 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 backwards compatibility, if `\patterns` is missing, the whole file is treated as a list of hyphenation patterns (only recognizing the `%` character as

the start of a comment).

If no `hpf` request is specified (either in the document or in a macro package), `gtrouff` won't hyphenate at all.

The `hpf` request appends a file of patterns to the current list.

The `hpfcode` request defines mapping values for character codes in hyphenation patterns. `hpf` or `hpf` then apply the mapping (after reading the patterns) before replacing or appending them to the current 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. You can use character codes that would be invalid otherwise. By default, everything maps to itself except letters 'A' to 'Z', which map to 'a' to 'z'.

The set of hyphenation patterns is associated with the current language set by the `hla` request. The `hpf` request is usually invoked by the `trouffrc` or `trouffrc-end` file; by default, `trouffrc` loads hyphenation patterns and exceptions for American English (in files `hyphen.us` and `hyphenex.us`).

A second call to `hpf` (for the same language) replaces the hyphenation patterns with the new ones.

Invoking `hpf` causes an error if there is no current hyphenation language.

`.hcode c1 code1 [c2 code2 ...]`

Set the hyphenation code of character *c1* to *code1*, that of *c2* to *code2*, etc. A hyphenation code must be a single input character (not a special character) other than a digit or a space.

To make hyphenation work, hyphenation codes must be set up. At start-up, `groff` only assigns hyphenation codes to the letters 'a'--'z' (mapped to themselves) and to the letters 'A'--'Z' (mapped to 'a'--'z'); all other hyphenation codes are set to zero. Normally, hyphenation patterns contain only lowercase letters, which should be applied regardless of case. In other words, the words 'FOO' and 'Foo' should be hyphenated exactly the same way as the word 'foo' is hyphenated, and this is what `hcode` is good for. Words that contain other letters won't be hyphenated properly if the corresponding hyphenation patterns actually do contain them. For example, the following `hcode` requests are necessary to assign hyphenation codes to the letters 'ÄäÖöÜüß' (this is needed for German):

```
.hcode ä ä Ä ä
.hcode ö ö Ö ö
.hcode ü ü Ü ü
.hcode ß ß
```

Without those assignments, `groff` treats German words like 'Kindergärten' (the plural form of 'kindergarten') as two substrings 'kinderg' and 'rten' because the hyphenation code of the umlaut a is zero by default. There is a German hyphenation pattern that covers 'kinder', so `groff` finds the hyphenation 'kin-der'. The other two hyphenation points ('kin-der-gär-ten') are missed.

This request is ignored if it has no parameter.

`.hym [length]`

`\n [.hym]`

Set the (right) hyphenation margin to *length*. If the current adjustment mode is not 'b'

or 'n', the line is not hyphenated if it is shorter than *length*. Without an argument, the hyphenation margin is reset to its default value, which is 0. The default scaling indicator for this request is 'm'. The hyphenation margin is associated with the current environment (see [Environments](#)).

A negative argument resets the hyphenation margin to zero, emitting a warning of type 'range'.

The current hyphenation margin is available in the `.hym` read-only number register.

`.hys` [*hyphenation\_space*]

`\n[.hys]`

Set the hyphenation space to *hyphenation\_space*. If the current adjustment mode is 'b' or 'n', don't hyphenate the line if it can be justified by adding no more than *hyphenation\_space* extra space to each word space. Without argument, the hyphenation space is set to its default value, which is 0. The default scaling indicator for this request is 'm'. The hyphenation space is associated with the current environment (see [Environments](#)).

A negative argument resets the hyphenation space to zero, emitting a warning of type 'range'.

The current hyphenation space is available in the `.hys` read-only number register.

`.shc` [*glyph*]

Set the *soft hyphen character* to *glyph*.<sup>14</sup> If the argument is omitted, the soft hyphen character is set to the default glyph `\(hy` (this is the start-up value of `gtrouff` also). The soft hyphen character is the glyph that is inserted when a word is hyphenated at a line break. If the soft hyphen character does not exist in the font of the character immediately preceding a potential break point, then the line is not broken at that point. Neither definitions (specified with the `char` request) nor translations (specified with the `tr` request) are considered when finding the soft hyphen character.

`.hla` *language*

`\n[.hla]`

Set the current hyphenation language to the string *language*. Hyphenation exceptions specified with the `hw` request and hyphenation patterns specified with the `hpf` and `hpfA` requests are both associated with the current hyphenation language. The `hla` request is usually invoked by the `troffrc` or the `troffrc-end` files; `troffrc` sets the default language to 'us'.

The current hyphenation language is available as a string in the read-only number register 'hla'.

```
.ds curr_language \n[.hla]
*[curr_language]
=> us
```

## 5.9. Manipulating Spacing

`.sp` [*distance*]

Space downwards *distance*. With no argument it advances 1 line. A negative argument causes `gtrouff` to move up the page the specified distance. If the argument is preceded by a '|' then `gtrouff` moves that distance from the top of the page. This

<sup>14</sup> *Soft hyphen character* is a misnomer since it is an output glyph.

request causes a line break, and that adds the current line spacing to the space you have just specified. The default scaling indicator is 'v'.

For convenience you may wish to use the following macros to set the height of the next line at a given distance from the top or the bottom of the page:

```
.de y-from-top-down
. sp | \ $1 - \ n [.v] u
..
.
.de y-from-bot-up
. sp | \ n [.p] u - \ $1 - \ n [.v] u
..
```

A call to 'y-from-bot-up 10c' means that the bottom of the next line will be at 10 cm from the paper edge at the bottom.

If a vertical trap is sprung during execution of `sp`, the amount of vertical space after the trap is discarded. For example, this

```
.de xxx
..
.
.wh 0 xxx
.
.pl 5v
foo
.sp 2
bar
.sp 50
baz
```

results in

```
foo

bar

baz
```

The amount of discarded space is available in the number register `.trunc`.

To protect `sp` against vertical traps, use the `vpt` request:

```
.vpt 0
.sp -3
.vpt 1
```

`.ls [nnn]`

`\n[.L]`

Output *nnn*-1 blank lines after each line of text. With no argument, `gtroff` uses the previous value before the last `ls` call.

```
.ls 2 \" This causes double-spaced output
.ls 3 \" This causes triple-spaced output
.ls \" Again double-spaced
```

The line spacing is associated with the current environment (see [Environments](#)).

The read-only number register `.L` contains the current line spacing setting.

See [Changing Type Sizes](#), for the requests `vs` and `pvs` as alternatives to `ls`.

`\x'spacing'`

`\n[.a]`

Sometimes, extra vertical spacing is only needed occasionally, e.g. to allow space for a tall construct (like an equation). The `\x` escape does this. The escape is given a numerical argument, usually enclosed in quotes (like `\x'3p'`); the default scaling indicator is `'v'`. If this number is positive extra vertical space is inserted below the current line. A negative number adds space above. If this escape is used multiple times on the same line, the maximum of the values is used.

See [Escapes](#), for details on parameter delimiting characters. The `.a` read-only number register contains the most recent (nonnegative) extra vertical line space.

Using `\x` can be necessary in combination with the `\b` escape, as the following example shows.

```
This is a test with the \[rs]b escape.
.br
This is a test with the \[rs]b escape.
.br
This is a test with \b'xyz'\x'-1m'\x'1m'.
.br
This is a test with the \[rs]b escape.
.br
This is a test with the \[rs]b escape.
```

produces

```
This is a test with the \b escape.
This is a test with the \b escape.
 x
This is a test with y.
 z
This is a test with the \b escape.
This is a test with the \b escape.
```

`.ns`

`.rs`

`\n[.ns]`

Enable *no-space mode*. In this mode, spacing (either via `sp` or via blank lines) is disabled. The `bp` request to advance to the next page is also disabled, except if it is accompanied by a page number (see [Page Control](#), for more information). This mode ends when actual text is output or the `rs` request is encountered, which ends no-space mode. The read-only number register `.ns` is set to 1 as long as no-space mode is active.

This request is useful for macros that conditionally insert vertical space before the text starts (for example, a paragraph macro could insert some space except when it is the first paragraph after a section header).

## 5.10. Tabs and Fields

A tab character (ASCII char 9, EBCDIC char 5) causes a horizontal movement to the next tab stop (much like it did on a typewriter).

`\t`

This escape is a non-interpreted tab character. In copy mode (see [Copy-in Mode](#)), `\t` is the same as a real tab character.

`.ta [n1 n2 ... nn T r1 r2 ... rn]`

`\n[.tabs]`

Change tab stop positions. This request takes a series of tab specifiers as arguments (optionally divided into two groups with the letter 'T') that indicate where each tab stop is to be (overriding any previous settings).

Tab stops can be specified absolutely, i.e., as the distance from the left margin. For example, the following sets 6 tab stops every one inch.

```
.ta 1i 2i 3i 4i 5i 6i
```

Tab stops can also be specified using a leading '+', which means that the specified tab stop is set relative to the previous tab stop. For example, the following is equivalent to the previous example.

```
.ta 1i +1i +1i +1i +1i +1i
```

`gtroff` supports an extended syntax to specify repeat values after the 'T' mark (these values are always taken as relative) -- this is the usual way to specify tabs set at equal intervals. The following is, yet again, the same as the previous examples. It does even more since it defines an infinite number of tab stops separated by one inch.

```
.ta T 1i
```

Now we are ready to interpret the full syntax given at the beginning: Set tabs at positions  $n1$ ,  $n2$ , ...,  $nn$  and then set tabs at  $nn+r1$ ,  $nn+r2$ , ...,  $nn+rn$  and then at  $nn+rn+r1$ ,  $nn+rn+r2$ , ...,  $nn+rn+rn$ , and so on.

Example: '4c +6c T 3c 5c 2c' is equivalent to '4c 10c 13c 18c 20c 23c 28c 30c ...'.

The material in each tab column (i.e., the column between two tab stops) may be justified to the right or left or centered in the column. This is specified by appending 'R', 'L', or 'C' to the tab specifier. The default justification is 'L'. Example:

```
.ta 1i 2iC 3iR
```

Some notes:

- The default unit of the `ta` request is 'm'.
- A tab stop is converted into a non-breakable horizontal movement that can be neither stretched nor squeezed. For example,

```
.ds foo a\tb\tc
.ta T 5i
\[foo]
```

creates a single line, which is a bit longer than 10 inches (a string is used to show exactly where the tab characters are). Now consider the following:

```
.ds bar a\tb b\tc
```

```
.ta T 5i
*[bar]
```

gtroff first converts the tab stops of the line into unbreakable horizontal movements, then splits the line after the second ‘b’ (assuming a sufficiently short line length). Usually, this isn’t what the user wants.

- Superfluous tabs (i.e., tab characters that do not correspond to a tab stop) are ignored except the first one, which delimits the characters belonging to the last tab stop for right-justifying or centering. Consider the following example

```
.ds Z foo\tbar\tfoo
.ds ZZ foo\tbar\tfoobar
.ds ZZZ foo\tbar\tfoo\tbar
.ta 2i 4iR
*[Z]
.br
*[ZZ]
.br
*[ZZZ]
.br
```

which produces the following output:

```
foo bar foo
foo bar foobar
foo bar foobar
```

The first line right-justifies the second ‘foo’ relative to the tab stop. The second line right-justifies ‘foobar’. The third line finally right-justifies only ‘foo’ because of the additional tab character, which marks the end of the string belonging to the last defined tab stop.

- Tab stops are associated with the current environment (see [Environments](#)).
- Calling `ta` without an argument removes all tab stops.
- The start-up value of `gtroff` is ‘T 0.5i’.

The read-only number register `.tabs` contains a string representation of the current tab settings suitable for use as an argument to the `ta` request.

```
.ds tab-string \n[.tabs]
*[tab-string]
=> T120u
```

The `troff` version of the Plan 9 operating system uses register `.s` for the same purpose.

#### `.tc` *[fill-glyph]*

Normally `gtroff` fills the space to the next tab stop with whitespace. This can be changed with the `tc` request. With no argument `gtroff` reverts to using whitespace, which is the default. The value of this *tab repetition character* is associated with the current environment (see [Environments](#)).<sup>15</sup>

<sup>15</sup> *Tab repetition character* is a misnomer since it is an output glyph.

```
.linetabs n
\n[.linetabs]
```

If *n* is missing or not zero, enable *line-tabs* mode, or disable it otherwise (the default). In line-tabs mode, `gtrouff` computes tab distances relative to the (current) output line instead of the input line.

For example, the following code:

```
.ds x a\t\c
.ds y b\t\c
.ds z c
.ta 1i 3i
*x
*y
*z
```

in normal mode, results in the output

```
a b c
```

in line-tabs mode, the same code outputs

```
a b c
```

Line-tabs mode is associated with the current environment. The read-only register `.linetabs` is set to 1 if in line-tabs mode, and 0 in normal mode.

### 5.10.1. Leaders

Sometimes it may be desirable to use the `tc` request to fill a particular tab stop with a given glyph (for example dots in a table of contents), but also normal tab stops on the rest of the line. For this `gtrouff` provides an alternate tab mechanism, called *leaders*, which does just that.

A leader character (character code 1) behaves similarly to a tab character: It moves to the next tab stop. The only difference is that for this movement, the fill glyph defaults to a period character and not to space.

```
\a
```

This escape is a non-interpreted leader character. In copy mode (see [Copy-in Mode](#)), `\a` is the same as a real leader character.

```
.lc [fill-glyph]
```

Declare the *leader repetition character*.<sup>16</sup> Without an argument, leaders act the same as tabs (i.e., using whitespace for filling). `gtrouff`'s start-up value is a dot ('.'). The value of the leader repetition character is associated with the current environment (see [Environments](#)).

For a table of contents, to name an example, tab stops may be defined so that the section number is one tab stop, the title is the second with the remaining space being filled with a line of dots, and then the page number slightly separated from the dots.

```
.ds entry 1.1\tFoo\a\t12
.lc .
.ta 1i 5i +.25i
*[entry]
```

<sup>16</sup> *Leader repetition character* is a misnomer since it is an output glyph.

This produces

1.1 Foo..... 12

### 5.10.2. Fields

*Fields* are a more general way of laying out tabular data. A field is defined as the data between a pair of *delimiting characters*. It contains substrings that are separated by *padding characters*. The width of a field is the distance on the *input* line from the position where the field starts to the next tab stop. A padding character inserts stretchable space similar to T<sub>E</sub>X's `\hss` command (thus it can even be negative) to make the sum of all substring lengths plus the stretchable space equal to the field width. If more than one padding character is inserted, the available space is evenly distributed among them.

`.fc [delim-char [padding-char]]`

Define a delimiting and a padding character for fields. If the latter is missing, the padding character defaults to a space character. If there is no argument at all, the field mechanism is disabled (which is the default). Note that contrary to e.g. the tab repetition character, delimiting and padding characters are *not* associated to the current environment (see [Environments](#)).

Example:

```
.fc # ^
.ta T 3i
#foo^bar^smurf#
.br
#foo^^bar^smurf#
```

and here the result:

```
foo bar smurf
foo bar smurf
```

### 5.11. Character Translations

The control character (‘.’) and the no-break control character (‘”) can be changed with the `cc` and `c2` requests, respectively.

`.cc [c]`

Set the control character to *c*. With no argument the default control character ‘.’ is restored. The value of the control character is associated with the current environment (see [Environments](#)).

`.c2 [c]`

Set the no-break control character to *c*. With no argument the default control character “” is restored. The value of the no-break control character is associated with the current environment (see [Environments](#)).

See [Requests](#). `.eo`

Disable the escape mechanism completely. After executing this request, the backslash character ‘\’ no longer starts an escape sequence.

This request can be very helpful in writing macros since it is not necessary then to double the escape character. Here an example:

```
.\ " This is a simplified version of the
```

```

.\" .BR request from the man macro package
.eo
.de BR
. ds result \&
. while (\n[.$] >= 2) \{\
. as result \fB\${1}\fR\${2}
. shift 2
. \}
. if \n[.$] .as result \fB\${1}
*[result]
. ft R
..
.ec

```

`.ec` [*c*]

Set the escape character to *c*. With no argument the default escape character `\` is restored. It can be also used to re-enable the escape mechanism after an `eo` request.

Note that changing the escape character globally likely breaks macro packages since `gtrouff` has no mechanism to ‘intern’ macros, i.e., to convert a macro definition into an internal form that is independent of its representation (TEX has this mechanism). If a macro is called, it is executed literally.

`.ecs`

`.ecr`

The `ecs` request saves the current escape character in an internal register. Use this request in combination with the `ec` request to temporarily change the escape character.

The `ecr` request restores the escape character saved with `ecs`. Without a previous call to `ecs`, this request sets the escape character to `\`.

`\\`

`\e`

`\E`

Print the current escape character (which is the backslash character `\` by default).

`\` is a ‘delayed’ backslash; more precisely, it is the default escape character followed by a backslash, which no longer has special meaning due to the leading escape character. It is *not* an escape sequence in the usual sense! In any unknown escape sequence `\X` the escape character is ignored and *X* is printed. But if *X* is equal to the current escape character, no warning is emitted.

As a consequence, only at top-level or in a diversion a backslash glyph is printed; in copy-in mode, it expands to a single backslash, which then combines with the following character to an escape sequence.

The `\E` escape differs from `\e` by printing an escape character that is not interpreted in copy mode. Use this to define strings with escapes that work when used in copy mode (for example, as a macro argument). The following example defines strings to begin and end a superscript:

```

.ds { \v'-.3m' \s' \En[.s]*60/100'
.ds } \s0\v'.3m'

```

Another example to demonstrate the differences between the various escape sequences, using a strange escape character, '-':

```
.ec -
.de xxx
--A'foo'
..
.xxx
=> -A'foo'
```

The result is surprising for most users, expecting '1' since 'foo' is a valid identifier. What has happened? As mentioned above, the leading escape character makes the following character ordinary. Written with the default escape character the sequence '--' becomes '\-' -- this is the minus sign.

If the escape character followed by itself is a valid escape sequence, only \E yields the expected result:

```
.ec -
.de xxx
-EA'foo'
..
.xxx
=> 1
```

\.

Similar to \, the sequence \. isn't a real escape sequence. As before, a warning message is suppressed if the escape character is followed by a dot, and the dot itself is printed.

```
.de foo
. nop foo
.
. de bar
. nop bar
\..
.
..
.foo
.bar
=> foo bar
```

The first backslash is consumed while the macro is read, and the second is swallowed while executing macro `foo`.

A *translation* is a mapping of an input character to an output glyph. The mapping occurs at output time, i.e., the input character gets assigned the metric information of the mapped output character right before input tokens are converted to nodes (see [gtroff Internals](#), for more on this process).

```
.tr abcd...
.trin abcd...
```

Translate character *a* to glyph *b*, character *c* to glyph *d*, etc. If there is an odd number of arguments, the last one is translated to an unstretchable space ('\ ').

The `trin` request is identical to `tr`, but when you unformat a diversion with

`asciify` it ignores the translation. See [Diversions](#), for details about the `asciify` request.

Some notes:

- Special characters (`\(xx`, `\[xxx]`, `\C'xxx'`, `\'`, `\'`, `\-`, `\_`), glyphs defined with the `char` request, and numbered glyphs (`\N'xxx'`) can be translated also.
- The `\e` escape can be translated also.
- Characters can be mapped onto the `\%` and `\~` escapes (but `\%` and `\~` can't be mapped onto another glyph).
- The following characters can't be translated: space (with one exception, see below), backspace, newline, leader (and `\a`), tab (and `\t`).
- Translations are not considered for finding the soft hyphen character set with the `shc` request.
- The pair `'c&'` (this is an arbitrary character `c` followed by the zero width space character) maps this character to nothing.

```
.tr a\&
foo bar
=> foo br
```

It is even possible to map the space character to nothing:

```
.tr aa \&
foo bar
=> foobar
```

As shown in the example, the space character can't be the first character/glyph pair as an argument of `tr`. Additionally, it is not possible to map the space character to any other glyph; requests like `'tr aa x'` undo `'tr aa \&'` instead.

If justification is active, lines are justified in spite of the 'empty' space character (but there is no minimal distance, i.e. the space character, between words).

- After an output glyph has been constructed (this happens at the moment immediately before the glyph is appended to an output glyph list, either by direct output, in a macro, diversion, or string), it is no longer affected by `tr`.
- Translating character to glyphs where one of them or both are undefined is possible also; `tr` does not check whether the entities in its argument do exist.

See [gtroff Internals](#).

- `troff` no longer has a hard-coded dependency on Latin-1; all `charXXX` entities have been removed from the font description files. This has a notable consequence that shows up in warnings like `can't find character with input code XXX` if the `tr` request isn't handled properly.

Consider the following translation:

```
.tr éÉ
```

This maps input character é onto glyph É, which is identical to glyph char201. But this glyph intentionally doesn't exist! Instead, \[char201] is treated as an input character entity and is by default mapped onto \['E], and gtrouff doesn't handle translations of translations.

The right way to write the above translation is

```
.tr é\['E]
```

In other words, the first argument of tr should be an input character or entity, and the second one a glyph entity.

- Without an argument, the tr request is ignored.

```
.trnt abcd..
```

trnt is the same as the tr request except that the translations do not apply to text that is transparently throughput into a diversion with \!. See [Diversions](#), for more information.

For example,

```
.tr ab
.di x
\!.tm a
.di
.x
```

prints 'b' to the standard error stream; if trnt is used instead of tr it prints 'a'.

## 5.12. Troff and Nroff Mode

Originally, nroff and troff were two separate programs, the former for TTY output, the latter for everything else. With GNUtroff, both programs are merged into one executable, sending its output to a device driver (grotty for TTY devices, grops for POSTSCRIPT, etc.) which interprets the intermediate output of gtrouff. For UNIX troff it makes sense to talk about *Nroff mode* and *Troff mode* since the differences are hardcoded. For GNU troff, this distinction is not appropriate because gtrouff simply takes the information given in the font files for a particular device without handling requests specially if a TTY output device is used.

Usually, a macro package can be used with all output devices. Nevertheless, it is sometimes necessary to make a distinction between TTY and non-TTY devices: gtrouff provides two built-in conditions 'n' and 't' for the if, ie, and while requests to decide whether gtrouff shall behave like nroff or like troff.

```
.troff
```

Make the 't' built-in condition true (and the 'n' built-in condition false) for if, ie, and while conditional requests. This is the default if gtrouff (*not* groff) is started with the -R switch to avoid loading of the start-up files troffrc and troffrc-end. Without -R, gtrouff stays in troff mode if the output device is not a TTY (e.g. 'ps').

```
.nroff
```

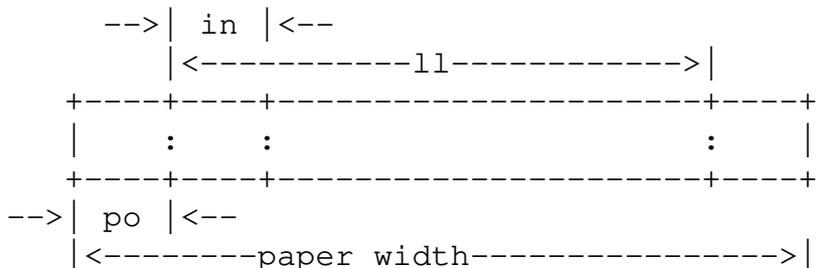
Make the 'n' built-in condition true (and the 't' built-in condition false) for if, ie, and while conditional requests. This is the default if gtrouff uses a TTY output device; the code for switching to nroff mode is in the file tty.tmac, which is loaded by the

start-up file `troffrc`.

See [Conditionals and Loops](#), for more details on built-in conditions.

### 5.13. Line Layout

The following drawing shows the dimensions that `gtroff` uses for placing a line of output onto the page. They are labeled with the request that manipulates each dimension.



These dimensions are:

- `po`      *Page offset* -- this is the leftmost position of text on the final output, defining the *left margin*.
- `in`      *Indentation* -- this is the distance from the left margin where text is printed.
- `ll`      *Line length* -- this is the distance from the left margin to right margin.

A simple demonstration:

```

.ll 3i
This is text without indentation.
The line length has been set to 3~inch.
.in +.5i
.ll -.5i
Now the left and right margins are both increased.
.in
.ll
Calling .in and .ll without parameters restore
the previous values.

```

Result:

```

This is text without indenta-
tion. The line length has
been set to 3 inch.
 Now the left and
 right margins are
 both increased.
Calling .in and .ll without
parameters restore the previ-
ous values.

```

```

.po [offset]
.po +offset
.po -offset
\n[.o]

```

Set horizontal page offset to *offset* (or increment or decrement the current value by

*offset*). Note that this request does not cause a break, so changing the page offset in the middle of text being filled may not yield the expected result. The initial value is 1 i. For TTY output devices, it is set to 0 in the startup file `troffrc`; the default scaling indicator is 'm' (and not 'v' as incorrectly documented in the original UNIX troff manual).

The current page offset can be found in the read-only number register '.o'.

If `po` is called without an argument, the page offset is reset to the previous value before the last call to `po`.

```
.po 3i
\n[.o]
=> 720

.po -1i
\n[.o]
=> 480

.po
\n[.o]
=> 720
```

`.in` [*indent*]

`.in` +*indent*

`.in` -*indent*

`\n[.i]`

Set indentation to *indent* (or increment or decrement the current value by *indent*). This request causes a break. Initially, there is no indentation.

If `in` is called without an argument, the indentation is reset to the previous value before the last call to `in`. The default scaling indicator is 'm'.

The indentation is associated with the current environment (see [Environments](#)).

If a negative indentation value is specified (which is not allowed), `gtroff` emits a warning of type 'range' and sets the indentation to zero.

The effect of `in` is delayed until a partially collected line (if it exists) is output. A temporary indentation value is reset to zero also.

The current indentation (as set by `in`) can be found in the read-only number register '.i'.

`.ti` *offset*

`.ti` +*offset*

`.ti` -*offset*

`\n[.in]`

Temporarily indent the next output line by *offset*. If an increment or decrement value is specified, adjust the temporary indentation relative to the value set by the `in` request.

This request causes a break; its value is associated with the current environment (see [Environments](#)). The default scaling indicator is 'm'. A call of `ti` without an argument is ignored.

If the total indentation value is negative (which is not allowed), `gtroff` emits a warning of type 'range' and sets the temporary indentation to zero. 'Total indentation' is either *offset* if specified as an absolute value, or the temporary plus normal

indentation, if *offset* is given as a relative value.

The effect of `ti` is delayed until a partially collected line (if it exists) is output.

The read-only number register `.in` is the indentation that applies to the current output line.

The difference between `.i` and `.in` is that the latter takes into account whether a partially collected line still uses the old indentation value or a temporary indentation value is active.

```
.ll [length]
```

```
.ll +length
```

```
.ll -length
```

```
\n[.l]
```

```
\n[.ll]
```

Set the line length to *length* (or increment or decrement the current value by *length*). Initially, the line length is set to 6.5i. The effect of `ll` is delayed until a partially collected line (if it exists) is output. The default scaling indicator is 'm'.

If `ll` is called without an argument, the line length is reset to the previous value before the last call to `ll`. If a negative line length is specified (which is not allowed), `gtroff` emits a warning of type 'range' and sets the line length to zero.

The line length is associated with the current environment (see [Environments](#)).

The current line length (as set by `ll`) can be found in the read-only number register `.l`. The read-only number register `.ll` is the line length that applies to the current output line.

Similar to `.i` and `.in`, the difference between `.l` and `.ll` is that the latter takes into account whether a partially collected line still uses the old line length value.

## 5.14. Line Control

It is important to understand how `gtroff` handles input and output lines.

Many escapes use positioning relative to the input line. For example, this

```
This is a \h'|1.2i'test.
```

```
This is a
\h'|1.2i'test.
```

produces

```
This is a test.
```

```
This is a test.
```

The main usage of this feature is to define macros that act exactly at the place where called.

```
.\ " A simple macro to underline a word
.de underline
. nop \ $1\1'|0\[u1]'
..
```

In the above example, `'|0'` specifies a negative distance from the current position (at the

end of the just emitted argument `\$1`) back to the beginning of the input line. Thus, the `\l` escape draws a line from right to left.

`gtroff` makes a difference between input and output line continuation; the latter is also called *interrupting* a line.

```
\RET,
```

```
\c,
```

```
\n[.int]
```

Continue a line. `\RET` (this is a backslash at the end of a line immediately followed by a newline) works on the input level, suppressing the effects of the following newline in the input.

```
This is a \
.test
=> This is a .test
```

The `|` operator is also affected.

`\c` works on the output level. Anything after this escape on the same line is ignored except `\R`, which works as usual. Anything before `\c` on the same line is appended to the current partial output line. The next non-command line after an interrupted line counts as a new input line.

The visual results depend on whether no-fill mode is active.

- If no-fill mode is active (using the `nf` request), the next input text line after `\c` is handled as a continuation of the same input text line.

```
.nf
This is a \c
test.
=> This is a test.
```

- If fill mode is active (using the `fi` request), a word interrupted with `\c` is continued with the text on the next input text line, without an intervening space.

```
This is a te\c
st.
=> This is a test.
```

Note that an intervening control line that causes a break is stronger than `\c`, flushing out the current partial line in the usual way.

The `.int` register contains a positive value if the last output line was interrupted with `\c`; this is associated with the current environment (see [Environments](#)).

## 5.15. Page Layout

`gtroff` provides some very primitive operations for controlling page layout.

```
.pl [length]
```

```
.pl +length
```

```
.pl -length
```

```
\n[.p]
```

Set the *page length* to *length* (or increment or decrement the current value by *length*). This is the length of the physical output page. The default scaling indicator is 'v'.

The current setting can be found in the read-only number register '.p'.

Note that this only specifies the size of the page, not the top and bottom margins. Those are not set by `gtroff` directly. See [Traps](#), for further information on how to do this.

Negative `p1` values are possible also, but not very useful: No trap is sprung, and each line is output on a single page (thus suppressing all vertical spacing).

If no argument or an invalid argument is given, `p1` sets the page length to 11 i.

`gtroff` provides several operations that help in setting up top and bottom titles (or headers and footers).

`.t1 'left' center' right'`

Print a *title line*. It consists of three parts: a left justified portion, a centered portion, and a right justified portion. The argument separator "" can be replaced with any character not occurring in the title line. The '%' character is replaced with the current page number. This character can be changed with the `pc` request (see below).

Without argument, `t1` is ignored.

Some notes:

- The line length set by the `l1` request is not honoured by `t1`; use the `lt` request (described below) instead, to control line length for text set by `t1`.
- A title line is not restricted to the top or bottom of a page.
- `t1` prints the title line immediately, ignoring a partially filled line (which stays untouched).
- It is not an error to omit closing delimiters. For example, `.t1 /foo'` is equivalent to `.t1 /foo//'`: It prints a title line with the left justified word 'foo'; the centered and right justified parts are empty.
- `t1` accepts the same parameter delimiting characters as the `\A` escape; see [Escapes](#).

`.lt [length]`

`.lt +length`

`.lt -length`

`\n[.lt]`

The title line is printed using its own line length, which is specified (or incremented or decremented) with the `lt` request. Initially, the title line length is set to 6.5i. If a negative line length is specified (which is not allowed), `gtroff` emits a warning of type 'range' and sets the title line length to zero. The default scaling indicator is 'm'. If `lt` is called without an argument, the title length is reset to the previous value before the last call to `lt`.

The current setting of this is available in the `.lt` read-only number register; it is associated with the current environment (see [Environments](#)).

```
.pn page
.pn +page
.pn -page
\n[.pn]
```

Change (increase or decrease) the page number of the *next* page. The only argument is the page number; the request is ignored without a parameter.

The read-only number register `.pn` contains the number of the next page: either the value set by a `pn` request, or the number of the current page plus 1.

```
.pc [char]
Change the page number character (used by the tl request) to a different character. With no argument, this mechanism is disabled. Note that this doesn't affect the number register %.
```

See [Traps](#).

## 5.16. Page Control

```
.bp [page]
.bp +page
.bp -page
\n[%]
```

Stop processing the current page and move to the next page. This request causes a break. It can also take an argument to set (increase, decrease) the page number of the next page (which actually becomes the current page after `bp` has finished). The difference between `bp` and `pn` is that `pn` does not cause a break or actually eject a page. See [Page Layout](#).

```
.de newpage \" define macro
'bp \" begin page
' sp .5i \" vertical space
.tl 'left top'center top'right top' \" title
' sp .3i \" vertical space
.. \" end macro
```

`bp` has no effect if not called within the top-level diversion (see [Diversions](#)).

The read-write register `%` holds the current page number.

The number register `.pe` is set to 1 while `bp` is active. See [Page Location Traps](#).

```
.ne [space]
```

It is often necessary to force a certain amount of space before a new page occurs. This is most useful to make sure that there is not a single *orphan* line left at the bottom of a page. The `ne` request ensures that there is a certain distance `,` specified by the first argument, before the next page is triggered (see [Traps](#), for further information). The default scaling indicator for `ne` is 'v'; the default value of *space* is 1 v if no argument is given.

For example, to make sure that no fewer than 2 lines get orphaned, do the following before each paragraph:

```
.ne 2
text text text
```

`ne` then automatically causes a page break if there is space for one line only.

`.sv [space]`

`.os`

`sv` is similar to the `ne` request; it reserves the specified amount of vertical space. If the desired amount of space exists before the next trap (or the bottom page boundary if no trap is set), the space is output immediately (ignoring a partially filled line, which stays untouched). If there is not enough space, it is stored for later output via the `os` request. The default value is 1 v if no argument is given; the default scaling indicator is 'v'.

Both `sv` and `os` ignore no-space mode. While the `sv` request allows negative values for *space*, `os` ignores them.

`\n[nl]`

This register contains the current vertical position. If the vertical position is zero and the top of page transition hasn't happened yet, `nl` is set to negative value. `gtroff` itself does this at the very beginning of a document before anything has been printed, but the main usage is to plant a header trap on a page if this page has already started.

Consider the following:

```
.de xxx
. sp
. tl "Header"
. sp
..
.
First page.
.bp
.wh 0 xxx
.nr nl (-1)
Second page.
```

Result:

```
First page.
```

```
...
```

```
Header
```

```
Second page.
```

```
...
```

Without resetting `nl` to a negative value, the just planted trap would be active beginning with the *next* page, not the current one.

See [Diversions](#), for a comparison with the `.h` and `.d` registers.

## 5.17. Fonts and Symbols

`gtroff` can switch fonts at any point in the text.

The basic set of fonts is ‘R’, ‘I’, ‘B’, and ‘BI’. These are Times Roman, Italic, Bold, and Bold Italic. For non-TTY devices, there is also at least one symbol font that contains various special symbols (Greek, mathematics).

### 5.17.1. Changing Fonts

```
.ft [font]
\ff
\f(fn)
\f[font]
\n[.sty]
```

The `ft` request and the `\f` escape change the current font to *font* (one-character name *f*, two-character name *fn*).

If *font* is a style name (as set with the `sty` request or with the `styles` command in the `DESC` file), use it within the current font family (as set with the `fam` request, `\F` escape, or with the `family` command in the `DESC` file).

It is not possible to switch to a font with the name ‘DESC’ (whereas this name could be used as a style name; however, this is not recommended).

With no argument or using ‘P’ as an argument, `.ft` switches to the previous font. Use `\f[]` to do this with the escape. The old syntax forms `\fP` or `\f[P]` are also supported.

Fonts are generally specified as upper-case strings, which are usually 1 to 4 characters representing an abbreviation or acronym of the font name. This is no limitation, just a convention.

The example below produces two identical lines.

```
eggs, bacon,
.ft B
spam
.ft
and sausage.
```

```
eggs, bacon, \fBspam\fP and sausage.
```

Note that `\f` doesn’t produce an input token in `gtroff`. As a consequence, it can be used in requests like `mc` (which expects a single character as an argument) to change the font on the fly:

```
.mc \f[I]x\f[]
```

The current style name is available in the read-only number register ‘.sty’ (this is a string-valued register); if the current font isn’t a style, the empty string is returned. It is associated with the current environment.

See [Font Positions](#), for an alternative syntax.

```
.ftr f[g]
```

Translate font *f* to font *g*. Whenever a font named *f* is referred to in a `\f` escape sequence, in the `F` and `S` conditional operators, or in the `ft`, `ul`, `bd`, `cs`, `tkf`, `special`, `fspecial`, `fp`, or `sty` requests, font *g* is used. If *g* is missing or equal to *f* the translation is undone.

Note that it is not possible to chain font translations. Example:

```
.ftr XXX TR
.ftr XXX YYY
.ft XXX
=> warning: can't find font 'XXX'
```

```
.fzoom f [zoom]
```

```
\n[.zoom]
```

Set magnification of font *f* to factor *zoom*, which must be a non-negative integer multiple of 1/1000th. This request is useful to adjust the optical size of a font in relation to the others. In the example below, font CR is magnified by 10% (the zoom factor is thus 1.1).

```
.fam P
.fzoom CR 1100
.ps 12
Palatino and \f[CR]Courier\f[]
```

A missing or zero value of *zoom* is the same as a value of 1000, which means no magnification. *f* must be a real font name, not a style.

Note that the magnification of a font is completely transparent to troff; a change of the zoom factor doesn't cause any effect except that the dimensions of glyphs, (word) spaces, kerns, etc., of the affected font are adjusted accordingly.

The zoom factor of the current font is available in the read-only number register '.zoom', in multiples of 1/1000th. It returns zero if there is no magnification.

### 5.17.2. Font Families

Due to the variety of fonts available, `gtroff` has added the concept of *font families* and *font styles*. The fonts are specified as the concatenation of the font family and style. Specifying a font without the family part causes `gtroff` to use that style of the current family.

Currently, fonts for the devices `-Tps`, `-Tpdf`, `-Tdvi`, `-Tlj4`, `-Tlbp`, and the X11 fonts are set up to this mechanism. By default, `gtroff` uses the Times family with the four styles 'R', 'I', 'B', and 'BI'.

This way, it is possible to use the basic four fonts and to select a different font family on the command line (see [Options](#)).

```
.fam [family]
\n[.fam]
\Ff
\F(fm
\F[family]
\n[.fn]
```

Switch font family to *family* (one-character name *f*, two-character name *fm*). If no argument is given, switch back to the previous font family. Use `\F[]` to do this with the escape. Note that `\FP` doesn't work; it selects font family 'P' instead.

The value at start-up is 'T'. The current font family is available in the read-only number register '.fam' (this is a string-valued register); it is associated with the current environment.

```
spam,
```

```
.fam H \" helvetica family
spam, \" used font is family H + style R = HR
.ft B \" family H + style B = font HB
spam,
.fam T \" times family
spam, \" used font is family T + style B = TB
.ft AR \" font AR (not a style)
baked beans,
.ft R \" family T + style R = font TR
and spam.
```

Note that `\F` doesn't produce an input token in `gtroff`. As a consequence, it can be used in requests like `mc` (which expects a single character as an argument) to change the font family on the fly:

```
.mc \F[P]x\F[]
```

The `.fn` register contains the current *real font name* of the current font. This is a string-valued register. If the current font is a style, the value of `\n[.fn]` is the proper concatenation of family and style name.

#### `.sty n style`

Associate *style* with font position *n*. A font position can be associated either with a font or with a style. The current font is the index of a font position and so is also either a font or a style. If it is a style, the font that is actually used is the font which name is the concatenation of the name of the current family and the name of the current style. For example, if the current font is 1 and font position 1 is associated with style 'R' and the current font family is 'T', then font 'TR' is used. If the current font is not a style, then the current family is ignored. If the requests `cs`, `bd`, `tkf`, `uf`, or `fspecial` are applied to a style, they are instead applied to the member of the current family corresponding to that style.

*n* must be a non-negative integer value.

The default family can be set with the `-f` option (see [Options](#)). The `styles` command in the `DESC` file controls which font positions (if any) are initially associated with styles rather than fonts. For example, the default setting for `POSTSCRIPT` fonts

```
styles R I B BI
```

is equivalent to

```
.sty 1 R
.sty 2 I
.sty 3 B
.sty 4 BI
```

`fam` and `\F` always check whether the current font position is valid; this can give surprising results if the current font position is associated with a style.

In the following example, we want to access the `POSTSCRIPT` font `FooBar` from the font family `Foo`:

```
.sty \n[.fp] Bar
.fam Foo
=> warning: can't find font `FooR'
```

The default font position at start-up is 1; for the `POSTSCRIPT` device, this is associated

with style 'R', so `gtrouff` tries to open `FooR`.

A solution to this problem is to use a dummy font like the following:

```
.fp 0 dummy TR \" set up dummy font at position 0
.sty \n[.fp] Bar \" register style 'Bar'
.ft 0 \" switch to font at position 0
.fam Foo \" activate family 'Foo'
.ft Bar \" switch to font 'FooBar'
```

See [Font Positions](#).

### 5.17.3. Font Positions

For the sake of old phototypesetters and compatibility with old versions of `trouff`, `gtrouff` has the concept of font *positions*, on which various fonts are mounted.

```
.fp pos font [external-name]
\n[.f]
\n[.fp]
```

Mount font *font* at position *pos* (which must be a non-negative integer). This numeric position can then be referred to with font changing commands. When `gtrouff` starts it is using font position 1 (which must exist; position 0 is unused usually at start-up).

The current font in use, as a font position, is available in the read-only number register 'f'. This can be useful to remember the current font for later recall. It is associated with the current environment (see [Environments](#)).

```
.nr save-font \n[.f]
.ft B
... text text text ...
.ft \n[save-font]
```

The number of the next free font position is available in the read-only number register 'fp'. This is useful when mounting a new font, like so:

```
.fp \n[.fp] NEATOFONT
```

Fonts not listed in the `DESC` file are automatically mounted on the next available font position when they are referenced. If a font is to be mounted explicitly with the `fp` request on an unused font position, it should be mounted on the first unused font position, which can be found in the `.fp` register. Although `gtrouff` does not enforce this strictly, it is not allowed to mount a font at a position whose number is much greater (approx. 1000 positions) than that of any currently used position.

The `fp` request has an optional third argument. This argument gives the external name of the font, which is used for finding the font description file. The second argument gives the internal name of the font, which is used to refer to the font in `gtrouff` after it has been mounted. If there is no third argument then the internal name is used as the external name. This feature makes it possible to use fonts with long names in compatibility mode.

Both the `ft` request and the `\f` escape have alternative syntax forms to access font positions.

```
.ft nnn
\f n
\f(nn
```

`\f[nnn]`

Change the current font position to *nnn* (one-digit position *n*, two-digit position *nn*), which must be a non-negative integer.

If *nnn* is associated with a style (as set with the `sty` request or with the `styles` command in the `DESC` file), use it within the current font family (as set with the `fam` request, the `\F` escape, or with the `family` command in the `DESC` file).

```

this is font 1
.ft 2
this is font 2
.ft \" switch back to font 1
.ft 3
this is font 3
.ft
this is font 1 again

```

See [Changing Fonts](#), for the standard syntax form.

#### 5.17.4. Using Symbols

A *glyph* is a graphical representation of a *character*. While a character is an abstract entity containing semantic information, a glyph is something that can be actually seen on screen or paper. It is possible that a character has multiple glyph representation forms (for example, the character 'A' can be either written in a roman or an italic font, yielding two different glyphs); sometimes more than one character maps to a single glyph (this is a *ligature* -- the most common is 'fi').

A *symbol* is simply a named glyph. Within `gtrouff`, all glyph names of a particular font are defined in its font file. If the user requests a glyph not available in this font, `gtrouff` looks up an ordered list of *special fonts*. By default, the `POSTSCRIPT` output device supports the two special fonts 'SS' (slanted symbols) and 'S' (symbols) (the former is looked up 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. To install additional special fonts locally (i.e. for a particular font), use the `fspecial` request.

Here are the exact rules how `gtrouff` searches a given symbol:

- If the symbol has been defined with the `char` request, use it. This hides a symbol with the same name in the current font.
- Check the current font.
- If the symbol has been defined with the `fchar` request, use it.
- Check whether the current font has a font-specific list of special fonts; test all fonts in the order of appearance in the last `fspecial` call if appropriate.
- If the symbol has been defined with the `fschar` request for the current font, use it.
- Check all fonts in the order of appearance in the last `special` call.
- If the symbol has been defined with the `schar` request, use it.

- As a last resort, consult all fonts loaded up to now for special fonts and check them, starting with the lowest font number. Note that this can sometimes lead to surprising results since the `fonts` line in the `DESC` file often contains empty positions, which are filled later on. For example, consider the following:

```
fonts 3 0 0 FOO
```

This mounts font `f00` at font position 3. We assume that `FOO` is a special font, containing glyph `f00`, and that no font has been loaded yet. The line

```
.fspecial BAR BAZ
```

makes font `BAZ` special only if font `BAR` is active. We further assume that `BAZ` is really a special font, i.e., the font description file contains the `special` keyword, and that it also contains glyph `f00` with a special shape fitting to font `BAR`. After executing `fspecial`, font `BAR` is loaded at font position 1, and `BAZ` at position 2.

We now switch to a new font `XXX`, trying to access glyph `f00` that is assumed to be missing. There are neither font-specific special fonts for `XXX` nor any other fonts made special with the `special` request, so `gtruff` starts the search for special fonts in the list of already mounted fonts, with increasing font positions. Consequently, it finds `BAZ` before `FOO` even for `XXX`, which is not the intended behaviour.

See [Font Files](#), and [Special Fonts](#), for more details. The list of available symbols is device dependent; see the `groff_char(7)` man page for a complete list of all glyphs. For example, say

```
man -Tdvi groff_char > groff_char.dvi
```

for a list using the default DVI fonts (not all versions of the `man` program support the `-T` option). If you want to use an additional macro package to change the used fonts, `groff` must be called directly:

```
groff -Tdvi -mec -man groff_char.7 > groff_char.dvi
```

Glyph names not listed in `groff_char(7)` are derived algorithmically, using a simplified version of the Adobe Glyph List (AGL) algorithm, which is described in <https://github.com/adobe-type-tools/agl-aglfn>. The (frozen) set of glyph names that can't be derived algorithmically is called *groff glyph list (GGL)*.

- A glyph for Unicode character U+XXXX[X[X]], which is not a composite character is named `uXXXX[X[X]]`. `X` must be an uppercase hexadecimal digit. Examples: `u1234`, `u008E`, `u12DB8`. The largest Unicode value is `0x10FFFF`. There must be at least four `X` digits; if necessary, add leading zeroes (after the 'u'). No zero padding is allowed for character codes greater than `0xFFFF`. Surrogates (i.e., Unicode values greater than `0xFFFF` represented with character codes from the surrogate area `U+D800-U+DFFF`) are not allowed too.
- A glyph representing more than a single input character is named  
Example: `u0045_0302_0301`.

For simplicity, all Unicode characters that are composites must be decomposed maximally (this is normalization form D in the Unicode standard); for example, `u00CA_0301` is not a valid glyph name since `U+00CA` (LATIN CAPITAL LETTER E WITH CIRCUMFLEX) can be further decomposed into `U+0045` (LATIN CAPITAL LETTER E) and `U+0302` (COMBINING CIRCUMFLEX ACCENT). `u0045_0302_0301` is thus the glyph name for

U+1EBE, LATIN CAPITAL LETTER E WITH CIRCUMFLEX AND ACUTE.

- groff maintains a table to decompose all algorithmically derived glyph names that are composites itself. For example, u0100 (LATIN LETTER A WITH MACRON) is automatically decomposed into u0041\_0304. Additionally, a glyph name of the GGL is preferred to an algorithmically derived glyph name; groff also automatically does the mapping. Example: The glyph u0045\_0302 is mapped to ^E.
- glyph names of the GGL can't be used in composite glyph names; for example, ^E\_u0301 is invalid.

`\(nm`

`\[name]`

`\[component1 component2 ...]`

Insert a symbol *name* (two-character name *nm*) or a composite glyph with component glyphs *component1*, *component2*, natural form '*n*' would collide with escapes.<sup>17</sup>

If *name* is undefined, a warning of type 'char' is generated, and the escape is ignored. See [Debugging](#), for information about warnings.

groff resolves `\[ . . . ]` with more than a single component as follows:

- Any component that is found in the GGL is converted to the uXXXX form.
- Any component uXXXX that is found in the list of decomposable glyphs is decomposed.
- The resulting elements are then concatenated with '\_' in between, dropping the leading 'u' in all elements but the first.

No check for the existence of any component (similar to `tr` request) is done.

Examples:

`\[A ho]` final glyph name would be u0041\_02DB. Note this is not the expected result: The ogonek glyph 'ho' is a spacing ogonek, but for a proper composite a non-spacing ogonek (U+0328) is necessary. Looking into the file `composite.tmac` one can find '`.composite ho u0328`', which changes the mapping of 'ho' while a composite glyph name is constructed, causing the final glyph name to be u0041\_0328.

`\[^E u0301]`

`\[^E aa]`

`\[E a^ aa]`

`\[E ^ ']`

u0045\_0302\_0301 in all forms (assuming proper calls of the `composite` request).

<sup>17</sup> Note that a one-character symbol is not the same as an input character, i.e., the character `a` is not the same as `\[a]`. By default, `groff` defines only a single one-character symbol, `\[-]`; it is usually accessed as `\-`. On the other hand, `gtrouff` has the special feature that `\[charXXX]` is the same as the input character with character code `XXX`. For example, `\[char97]` is identical to the letter `a` if ASCII encoding is active.

It is not possible to define glyphs with names like ‘A ho’ within a groff font file. This is not really a limitation; instead, you have to define `u0041_0328`.

### `\C'xxx'`

Typeset the glyph named `xxx`.<sup>18</sup> Normally it is more convenient to use `\[xxx]`, but `\C` has the advantage that it is compatible with newer versions of AT&T `ttruff` and is available in compatibility mode.

### `.composite from to`

Map glyph name *from* to glyph name *to* if it is used in `\[...]` with more than one component. See above for examples.

This mapping is based on glyph names only; no check for the existence of either glyph is done.

A set of default mappings for many accents can be found in the file `composite.tmac`, which is loaded at start-up.

### `\N'n'`

Typeset the glyph with code `n` in the current font (`n` is **not** the input character code). The number `n` can be any non-negative decimal integer. Most devices only have glyphs with codes between 0 and 255; the Unicode output device uses codes in the range 0--65535. If the current font does not contain a glyph with that code, special fonts are *not* searched. The `\N` escape sequence can be conveniently used in conjunction with the `char` request:

```
.char \[phone] \f[ZD]\N'37'
```

The code of each glyph is given in the fourth column in the font description file after the `charset` command. It is possible to include unnamed glyphs in the font description file by using a name of ‘---’; the `\N` escape sequence is the only way to use these.

No kerning is applied to glyphs accessed with `\N`.

Some escape sequences directly map onto special glyphs.

### `\'`

This is a backslash followed by the apostrophe character, ASCII character `0x27` (EBCDIC character `0x7D`). The same as `\[aa]`, the acute accent.

### `\``

This is a backslash followed by ASCII character `0x60` (EBCDIC character `0x79` usually). The same as `\[ga]`, the grave accent.

### `\-`

This is the same as `\[-]`, the minus sign in the current font.

### `\_`

This is the same as `\[ul]`, the underline character.

### `.cflags n c1 c2 ...`

Input characters and symbols have certain properties associated with it.<sup>19</sup> These properties can be modified with the `cflags` request. The first argument is the sum

<sup>18</sup> `\C` is actually a misnomer since it accesses an output glyph.

<sup>19</sup> Note that the output glyphs themselves don't have such properties. For `gtruff`, a glyph is a numbered box with a given width, depth, and height, nothing else. All manipulations with the `cflags` request work on the input level.

of the desired flags and the remaining arguments are the characters or symbols to have those properties. It is possible to omit the spaces between the characters or symbols. Instead of single characters or symbols you can also use character classes (see [Character Classes](#) for more details).

- 1        The character ends sentences (initially characters ‘.?!’ have this property).
- 2        Lines can be broken before the character (initially no characters have this property). This only works if both the characters before and after have non-zero hyphenation codes (as set with the `hcode` request). Use value 64 to override this behaviour.
- 4        Lines can be broken after the character (initially the character ‘-’ and the symbols ‘\[hy]’ and ‘\[em]’ have this property). This only works if both the characters before and after have non-zero hyphenation codes (as set with the `hcode` request). Use value 64 to override this behaviour.
- 8        The character overlaps horizontally if used as a horizontal line building element. Initially the symbols ‘\[ul]’, ‘\[rn]’, ‘\[ru]’, ‘\[radicalex]’, and ‘\[sqrtex]’ have this property.
- 16       The character overlaps vertically if used as vertical line building element. Initially symbol ‘\[br]’ has this property.
- 32       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; in other words the character is *transparent* for the purposes of end-of-sentence recognition -- this is the same as having a zero space factor in T<sub>E</sub>X (initially characters ‘”’)\*’ and the symbols ‘\[dg]’, ‘\[rq]’, and ‘\[cq]’ have this property).
- 64       Ignore hyphenation code values of the surrounding characters. Use this in combination with values 2 and 4 (initially no characters have this property). For example, if you need an automatic break point after the en-dash in number ranges like ‘3000--5000’, insert
 

```
.cflags 68 \ (en
```

 into your document. Note, however, that this can lead to bad layout if done without thinking; in most situations, a better solution instead of changing the `cflags` value is to insert `\:` right after the hyphen at the places that really need a break point.
- 128      Prohibit a line break before the character, but allow a line break after the character. This works only in combination with flags 256 and 512 (see below) and has no effect otherwise.
- 256      Prohibit a line break after the character, but allow a line break before the character. This works only in combination with flags 128 and 512 (see below) and has no effect otherwise.

512 Allow line break before or after the character. This works only in combination with flags 128 and 256 and has no effect otherwise.

Contrary to flag values 2 and 4, the flags 128, 256, and 512 work pairwise. If, for example, the left character has value 512, and the right character 128, no line break gets inserted. If we use value 6 instead for the left character, a line break after the character can't be suppressed since the right neighbour character doesn't get examined.

```
.char g [string]
.fchar g [string]
.fschar f g [string]
.schar g [string]
```

Define a new glyph *g* to be *string* (which can be empty).<sup>20</sup> Every time glyph *g* needs to be printed, *string* is processed in a temporary environment and the result is wrapped up into a single object. Compatibility mode is turned off and the escape character is set to `\` while *string* is being processed. Any emboldening, constant spacing or track kerning is applied to this object rather than to individual characters in *string*.

A glyph defined by these requests can be used just like a normal glyph provided by the output device. In particular, other characters can be translated to it with the `tr` or `trin` requests; it can be made the leader character by the `lc` request; repeated patterns can be drawn with the glyph using the `\l` and `\L` escape sequences; words containing the glyph can be hyphenated correctly if the `hcode` request is used to give the glyph's symbol a hyphenation code.

There is a special anti-recursion feature: Use of `g` within the glyph's definition is handled like normal characters and symbols not defined with `char`.

Note that the `tr` and `trin` requests take precedence if `char` accesses the same symbol.

```
.tr XY
X
 => Y
.char X Z
X
 => Y
.tr XX
X
 => Z
```

The `fchar` request defines a fallback glyph: `gtrouff` only checks for glyphs defined with `fchar` if it cannot find the glyph in the current font. `gtrouff` carries out this test before checking special fonts.

`fschar` defines a fallback glyph for font *f*: `gtrouff` checks for glyphs defined with `fschar` after the list of fonts declared as font-specific special fonts with the `fspecial` request, but before the list of fonts declared as global special fonts with the `special` request.

---

<sup>20</sup> `char` is a misnomer since an output glyph is defined.

Finally, the `schar` request defines a global fallback glyph: `gtroff` checks for glyphs defined with `schar` after the list of fonts declared as global special fonts with the `special` request, but before the already mounted special fonts.

See [Using Symbols](#), for a detailed description of the glyph searching mechanism in `gtroff`.

```
.rchar c1 c2 ...
```

```
.rfschar f c1 c2 ...
```

Remove the definitions of glyphs `c1`, `c2`, ... This undoes the effect of a `char`, `fchar`, or `schar` request.

It is possible to omit the whitespace between arguments.

The request `rfschar` removes glyph definitions defined with `fschar` for glyph `f`.

See [Special Characters](#).

### 5.17.5. Character Classes

Classes are particularly useful for East Asian languages such as Chinese, Japanese, and Korean, where the number of needed characters is much larger than in European languages, and where large sets of characters share the same properties.

```
.class n c1 c2 ...
```

In `groff`, a *character class* (or simply “class”) is a set of characters, grouped by some user aspect. The `class` request defines such classes so that other requests can refer to all characters belonging to this set with a single class name. Currently, only the `cflags` request can handle character classes.

A `class` request takes a class name followed by a list of entities. In its simplest form, the entities are characters or symbols:

```
.class [prepunct] , : ; > }
```

Since class and glyph names share the same namespace, it is recommended to start and end the class name with `[` and `]`, respectively, to avoid collisions with normal `groff` symbols (and symbols defined by the user). In particular, the presence of `]` in the symbol name intentionally prevents the usage of `\[...]`, thus you must use the `\C` escape to access a class with such a name.

You can also use a special character range notation, consisting of a start character or symbol, followed by `'-'`, and an end character or symbol. Internally, `gtroff` converts these two symbol names to Unicode values (according to the `groff` glyph gist), which then give the start and end value of the range. If that fails, the class definition is skipped.

Finally, classes can be nested, too.

Here is a more complex example:

```
.class [prepnctx] \C' [prepunct]' \[u2013]-\[u2016]
```

The class `'prepnctx'` now contains the contents of the class `prepunct` as defined above (the set `' : ; > '`), and characters in the range between `U+2013` and `U+2016`.

If you want to add `'-'` to a class, it must be the first character value in the argument list, otherwise it gets misinterpreted as a range.

Note that it is not possible to use class names within range definitions.

Typical use of the `class` request is to control line-breaking and hyphenation rules as defined by the `cflags` request. For example, to inhibit line breaks before the characters belonging to the `prepunctx` class, you can write:

```
.cflags 2 \C' [prepunctx]'
```

See the `cflags` request in [Using Symbols](#), for more details.

### 5.17.6. Special Fonts

Special fonts are those that `gtroff` searches when it cannot find the requested glyph in the current font. The Symbol font is usually a special font.

`gtroff` provides the following two requests to add more special fonts. See [Using Symbols](#), for a detailed description of the glyph searching mechanism in `gtroff`.

Usually, only non-TTY devices have special fonts.

```
.special [s1 s2 ...]
```

```
.fspecial f[s1 s2 ...]
```

Use the `special` request to define special fonts. Initially, this list is empty.

Use the `fspecial` request to designate special fonts only when font *f* is active. Initially, this list is empty.

Previous calls to `special` or `fspecial` are overwritten; without arguments, the particular list of special fonts is set to empty. Special fonts are searched in the order they appear as arguments.

All fonts that appear in a call to `special` or `fspecial` are loaded.

See [Using Symbols](#), for the exact search order of glyphs.

### 5.17.7. Artificial Fonts

There are a number of requests and escapes for artificially creating fonts. These are largely vestiges of the days when output devices did not have a wide variety of fonts, and when `nroff` and `troff` were separate programs. Most of them are no longer necessary in GNU `troff`. Nevertheless, they are supported.

```
\H'height
```

```
\H'+height'
```

```
\H'-height'
```

```
\n[.height]
```

Change (increment, decrement) the height of the current font, but not the width. If *height* is zero, restore the original height. Default scaling indicator is 'z'.

The read-only number register `.height` contains the font height as set by `\H`.

Currently, only the `-Tps` and `-Tpdf` devices support this feature.

Note that `\H` doesn't produce an input token in `gtroff`. As a consequence, it can be used in requests like `mc` (which expects a single character as an argument) to change the font on the fly:

```
.mc \H'+5z' x\H'0'
```

In compatibility mode, `gtroff` behaves differently: If an increment or decrement is used, it is always taken relative to the current point size and not relative to the previously selected font height. Thus,

```
.cp 1
 \H'+5'test \H'+5'test
```

prints the word 'test' twice with the same font height (five points larger than the current font size).

`\S'slant`

`\n[.slant]`

Slant the current font by *slant* degrees. Positive values slant to the right. Only integer values are possible.

The read-only number register `.slant` contains the font slant as set by `\S`.

Currently, only the `-Tps` and `-Tpdf` devices support this feature.

Note that `\S` doesn't produce an input token in `gtrouff`. As a consequence, it can be used in requests like `mc` (which expects a single character as an argument) to change the font on the fly:

```
.mc \S'20'x\S'0'
```

This request is incorrectly documented in the original UNIX `trouff` manual; the slant is always set to an absolute value.

`.ul [lines]`

The `ul` request normally underlines subsequent lines if a TTY output device is used. Otherwise, the lines are printed in italics (only the term 'underlined' is used in the following). The single argument is the number of input lines to be underlined; with no argument, the next line is underlined. If *lines* is zero or negative, stop the effects of `ul` (if it was active). Requests and empty lines do not count for computing the number of underlined input lines, even if they produce some output like `tl`. Lines inserted by macros (e.g. invoked by a trap) do count.

At the beginning of `ul`, the current font is stored and the underline font is activated. Within the span of a `ul` request, it is possible to change fonts, but after the last line affected by `ul` the saved font is restored.

This number of lines still to be underlined is associated with the current environment (see [Environments](#)). The underline font can be changed with the `uf` request.

The `ul` request does not underline spaces.

`.cu [lines]`

The `cu` request is similar to `ul` but underlines spaces as well (if a TTY output device is used).

`.uf font`

Set the underline font (globally) used by `ul` and `cu`. By default, this is the font at position 2. *font* can be either a non-negative font position or the name of a font.

`.bd font [offset]`

`.bd font1 font2 [offset]`

`\n[.b]`

Artificially create a bold font by printing each glyph twice, slightly offset.

Two syntax forms are available.

- Imitate a bold font unconditionally. The first argument specifies the font to embolden, and the second is the number of basic units, minus one, by which the two glyphs are offset. If the second argument is missing,

emboldening is turned off.

*font* can be either a non-negative font position or the name of a font.

*offset* is available in the `.b` read-only register if a special font is active; in the `bd` request, its default unit is 'u'.

- Imitate a bold form conditionally. Embolden *font1* by *offset* only if font *font2* is the current font. This command can be issued repeatedly to set up different emboldening values for different current fonts. If the second argument is missing, emboldening is turned off for this particular current font.

This affects special fonts only (either set up with the `special` command in font files or with the `fspecial` request).

`.cs font [width [em-size]]`

Switch to and from *constant glyph space mode*. If activated, the width of every glyph is *width/36* ems. The em size is given absolutely by *em-size*; if this argument is missing, the em value is taken from the current font size (as set with the `ps` request) when the font is effectively in use. Without second and third argument, constant glyph space mode is deactivated.

Default scaling indicator for *em-size* is 'z'; *width* is an integer.

### 5.17.8. Ligatures and Kerning

Ligatures are groups of characters that are run together, i.e, producing a single glyph. For example, the letters 'f' and 'i' can form a ligature 'fi' as in the word 'file'. This produces a cleaner look (albeit subtle) to the printed output. Usually, ligatures are not available in fonts for TTY output devices.

Most `POSTSCRIPT` fonts support the `fi` and `fl` ligatures. The C/A/T typesetter that was the target of AT&T `truff` also supported 'ff', 'ffi', and 'ffl' ligatures. Advanced typesetters or 'expert' fonts may include ligatures for 'ft' and 'ct', although GNU `truff` does not support these (yet).

Only 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.

`.lg [flag]`

`\n[.lg]`

Switch the ligature mechanism on or off; if the parameter is non-zero or missing, ligatures are enabled, otherwise disabled. Default is on. The current ligature mode can be found in the read-only number register `.lg` (set to 1 or 2 if ligatures are enabled, 0 otherwise).

Setting the ligature mode to 2 enables the two-character ligatures (`fi`, `fl`, and `ff`) and disables the three-character ligatures (`ffi` and `ffl`).

*Pairwise kerning* is another subtle typesetting mechanism that modifies the distance between a glyph pair to improve readability. In most cases (but not always) the distance is decreased. For example, compare the combination of the letters 'V' and 'A'. With kerning, 'VA' is printed. Without kerning it appears as 'VA'. Typewriter-like fonts and fonts for terminals where all glyphs have the same width don't use kerning.

`.kern [flag]`

`\n[.kern]`

Switch kerning on or off. If the parameter is non-zero or missing, enable pairwise kerning, otherwise disable it. The read-only number register `.kern` is set to 1 if pairwise kerning is enabled, 0 otherwise.

If the font description file contains pairwise kerning information, glyphs from that font are kerned. Kerning between two glyphs can be inhibited by placing `\&` between them: `'\&A'`.

See [Font File Format](#).

*Track kerning* expands or reduces the space between glyphs. This can be handy, for example, if you need to squeeze a long word onto a single line or spread some text to fill a narrow column. It must be used with great care since it is usually considered bad typography if the reader notices the effect.

`.tkf f s1 n1 s2 n2`

Enable track kerning for font *f*. If the current font is *f* the width of every glyph is increased by an amount between *n1* and *n2* (*n1*, *n2* can be negative); if the current point size is less than or equal to *s1* the width is increased by *n1*; if it is greater than or equal to *s2* the width is increased by *n2*; if the point 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 point size.

The default scaling indicator is 'z' for *s1* and *s2*, 'p' for *n1* and *n2*.

Note that the track kerning amount is added even to the rightmost glyph in a line; for large values it is thus recommended to increase the line length by the same amount to compensate it.

Sometimes, when typesetting letters of different fonts, more or less space at such boundaries is needed. There are two escapes to help with this.

`\/`

Increase the width of the preceding glyph so that the spacing between that glyph and the following glyph is correct if the following glyph is a roman glyph. For example, if an italic *f* is immediately followed by a roman right parenthesis, then in many fonts the top right portion of the *f* overlaps the top left of the right parenthesis. Use this escape sequence whenever an italic glyph is immediately followed by a roman glyph without any intervening space. This small amount of space is also called *italic correction*.

```
\f[I]f\f[R])
=> f)
\f[I]f\/\f[R])
=> f)
```

`\,`

Modify the spacing of the following glyph so that the spacing between that glyph and the preceding glyph is correct if the preceding glyph is a roman glyph. Use this escape sequence whenever a roman glyph is immediately followed by an italic glyph without any intervening space. In analogy to above, this space could be called *left italic correction*, but this term isn't used widely.

```
q\f[I]f
=> qf
q\,\f[I]f
=> qf
```

\&amp;

Insert a zero-width character, which is invisible. Its intended use is to stop interaction of a character with its surrounding.

- It prevents the insertion of extra space after an end-of-sentence character.

```
Test.
Test.
=> Test. Test.
Test.\&
Test.
=> Test. Test.
```

- It prevents interpretation of a control character at the beginning of an input line.

```
.Test
=> warning: `Test' not defined
\&.Test
=> .Test
```

- It prevents kerning between two glyphs.

```
VA
=> VA
V\&A
=> VA
```

- It is needed to map an arbitrary character to nothing in the `tr` request (see [Character Translations](#)).

\)

This escape is similar to `\&` except that it behaves like a character declared with the `cflags` request to be transparent for the purposes of an end-of-sentence character.

Its main usage is in macro definitions to protect against arguments starting with a control character.

```
.de xxx
)\$1
..
.de yyy
\&\$1
..
This is a test.\c
.xxx '
This is a test.
=>This is a test.' This is a test.
This is a test.\c
.yyy '
This is a test.
=>This is a test.' This is a test.
```

## 5.18. Sizes

`gtrouff` uses two dimensions with each line of text, type size and vertical spacing. The *type size* is approximately the height of the tallest glyph.<sup>21</sup> *Vertical spacing* is the amount of space `gtrouff` allows for a line of text; normally, this is about 20% larger than the current type size. Ratios smaller than this can result in hard-to-read text; larger than this, it spreads the text out more vertically (useful for term papers). By default, `gtrouff` uses 10 point type on 12 point spacing.

The difference between type size and vertical spacing is known, by typesetters, as *leading* (this is pronounced 'ledding').

### 5.18.1. Changing Type Sizes

```
.ps [size]
.ps +size
.ps -size
\s size
\n [.s]
```

Use the `.ps` request or the `\s` escape to change (increase, decrease) the type size (in points). Specify *size* as either an absolute point size, or as a relative change from the current size. The size 0 (for both `.ps` and `\s`), or no argument (for `.ps` only), goes back to the previous size.

Default scaling indicator of *size* is 'z'. If *size* is negative, it is set to 1 u.

The read-only number register `.s` returns the point size in points as a decimal fraction. This is a string. To get the point size in scaled points, use the `.ps` register instead.

`.s` is associated with the current environment (see [Environments](#)).

```
snap, snap,
.ps +2
grin, grin,
.ps +2
wink, wink, \s+2nudge, nudge, \s+8 say no more!
.ps 10
```

The `\s` escape may be called in a variety of ways. Much like other escapes there must be a way to determine where the argument ends and the text begins. Any of the following forms are valid:

|                   |                                                                                             |
|-------------------|---------------------------------------------------------------------------------------------|
| <code>\sn</code>  | Set the point size to <i>n</i> points. <i>n</i> must be either 0 or in the range 4 to 39.   |
| <code>\s+n</code> |                                                                                             |
| <code>\s-n</code> | Increase or decrease the point size by <i>n</i> points. <i>n</i> must be exactly one digit. |

---

<sup>21</sup> This is usually the parenthesis. Note that in most cases the real dimensions of the glyphs in a font are *not* related to its type size! For example, the standard `POSTSCRIPT` font families 'Times Roman', 'Helvetica', and 'Courier' can't be used together at 10pt; to get acceptable output, the size of 'Helvetica' has to be reduced by one point, and the size of 'Courier' must be increased by one point.

`\s (nn` Set the point size to *nn* points. *nn* must be exactly two digits.

`\s+ (nn`

`\s- (nn`

`\s (+nn`

`\s (-nn` Increase or decrease the point size by *nn* points. *nn* must be exactly two digits.

Note that `\s` doesn't produce an input token in `gtroff`. As a consequence, it can be used in requests like `mc` (which expects a single character as an argument) to change the font on the fly:

```
.mc \s[20]x\s[0]
```

See [Fractional Type Sizes](#), for yet another syntactical form of using the `\s` escape.

`.sizes s1 s2 ... sn [0]`

Some devices may only have certain permissible sizes, in which case `gtroff` rounds to the nearest permissible size. The `DESC` file specifies which sizes are permissible for the device.

Use the `sizes` request to change the permissible sizes for the current output device. Arguments are in scaled points; the `sizescale` line in the `DESC` file for the output device provides the scaling factor. For example, if the scaling factor is 1000, then the value 12000 is 12 points.

Each argument can be a single point size (such as '12000'), or a range of sizes (such as '4000-72000'). You can optionally end the list with a zero.

`.vs [space]`

`.vs +space`

`.vs -space`

`\n [.v]`

Change (increase, decrease) the vertical spacing by *space*. The default scaling indicator is 'p'.

If `vs` is called without an argument, the vertical spacing is reset to the previous value before the last call to `vs`.

`gtroff` creates a warning of type 'range' if *space* is negative; the vertical spacing is then set to smallest positive value, the vertical resolution (as given in the `.v` register).

Note that '`vs 0`' isn't saved in a diversion since it doesn't result in a vertical motion. You explicitly have to repeat this command before inserting the diversion.

The read-only number register `.v` contains the current vertical spacing; it is associated with the current environment (see [Environments](#)).

The effective vertical line spacing consists of four components. Breaking a line causes the following actions (in the given order).

- Move the current point vertically by the *extra pre-vertical line space*. This is the minimum value of all `\x` escapes with a negative argument in the current output line.

- Move the current point vertically by the vertical line spacing as set with the `vs` request.
- Output the current line.
- Move the current point vertically by the *extra post-vertical line space*. This is the maximum value of all `\x` escapes with a positive argument in the line that has just been output.
- Move the current point vertically by the *post-vertical line spacing* as set with the `pvs` request.

It is usually better to use `vs` or `pvs` instead of `ls` to produce double-spaced documents: `vs` and `pvs` have a finer granularity for the inserted vertical space compared to `ls`; furthermore, certain preprocessors assume single-spacing.

See [Manipulating Spacing](#), for more details on the `\x` escape and the `ls` request.

```
.pvs [space]
.pvs +space
.pvs -space
\n[.pvs]
```

Change (increase, decrease) the post-vertical spacing by *space*. The default scaling indicator is 'p'.

If `pvs` is called without an argument, the post-vertical spacing is reset to the previous value before the last call to `pvs`.

`gtroff` creates a warning of type 'range' if *space* is zero or negative; the vertical spacing is then set to zero.

The read-only number register `.pvs` contains the current post-vertical spacing; it is associated with the current environment (see [Environments](#)).

### 5.18.2. Fractional Type Sizes

A *scaled point* is equal to  $1/\text{sizescale}$  points, where *sizescale* is specified in the `DESC` file (1 by default). There is a new scale indicator 'z', which has the effect of multiplying by *sizescale*. Requests and escape sequences in `gtroff` interpret arguments that represent a point size as being in units of scaled points, but they evaluate each such argument using a default scale indicator of 'z'. Arguments treated in this way are the argument to the `ps` request, the third argument to the `cs` request, the second and fourth arguments to the `tkf` request, the argument to the `\H` escape sequence, and those variants of the `\s` escape sequence that take a numeric expression as their argument (see below).

For example, suppose *sizescale* is 1000; then a scaled point is equivalent to a millipoint; the request `.ps 10.25` is equivalent to `.ps 10.25z` and thus sets the point size to 10250 scaled points, which is equal to 10.25 points.

`gtroff` disallows the use of the 'z' scale indicator in instances where it would make no sense, such as a numeric expression whose default scale indicator was neither 'u' nor 'z'. Similarly it would make no sense to use a scaling indicator other than 'z' or 'u' in a numeric expression whose default scale indicator was 'z', and so `gtroff` disallows this as well.

There is also new scale indicator 's', which multiplies by the number of units in a scaled point. So, for example, `\n[.ps]s` is equal to `'1m'`. Be sure not to confuse the 's' and 'z'

scale indicators.

`\n[.ps]`

A read-only number register returning the point size in scaled points.

`.ps` is associated with the current environment (see [Environments](#)).

`\n[.psr]`

`\n[.sr]`

The last-requested point size in scaled points is contained in the `.psr` read-only number register. The last requested point size in points as a decimal fraction can be found in `.sr`. This is a string-valued read-only number register.

Note that the requested point sizes are device-independent, whereas the values returned by the `.ps` and `.s` registers are not. For example, if a point size of 11 pt is requested, and a `sizes` request (or a `sizescale` line in a `DESC` file) specifies 10.95pt instead, this value is actually used.

Both registers are associated with the current environment (see [Environments](#)).

The `\s` escape has the following syntax for working with fractional type sizes:

`\s[n]`

`\s' n'` Set the point size to  $n$  scaled points;  $n$  is a numeric expression with a default scale indicator of 'z'.

`\s[+n]`

`\s[-n]`

`\s+[n]`

`\s-[n]`

`\s'+n'`

`\s'-n'`

`\s+' n'`

`\s-' n'` Increase or decrease the point size by  $n$  scaled points;  $n$  is a numeric expression (which may start with a minus sign) with a default scale indicator of 'z'.

See [Font Files](#).

## 5.19. Strings

`gtrouff` has string variables, which are entirely for user convenience (i.e. there are no built-in strings except `.T`, but even this is a read-write string variable).

Although the following requests can be used to create strings, simply using an undefined string will cause it to be defined as empty. See [Identifiers](#).

`.ds name [string]`

`.ds1 name [string]`

`\*n`

`\*(nm`

`\*[name arg1 arg2 ...]`

Define and access a string variable *name* (one-character name  $n$ , two-character name  $nm$ ). If *name* already exists, `ds` overwrites the previous definition. Only the syntax form using brackets can take arguments that are handled identically to macro

arguments; the single exception is that a closing bracket as an argument must be enclosed in double quotes. See [Request and Macro Arguments](#), and [Parameters](#).

Example:

```
.ds foo a \ $1 test
.
This is *[foo nice].
=> This is a nice test.
```

The `\*` escape *interpolates* (expands in-place) a previously defined string variable. To be more precise, the stored string is pushed onto the input stack, which is then parsed by `gtrouff`. Similar to number registers, it is possible to nest strings, i.e., string variables can be called within string variables.

If the string named by the `\*` escape does not exist, it is defined as empty, and a warning of type 'mac' is emitted (see [Debugging](#), for more details).

`ds` request takes up the entire line including trailing spaces. This means that comments on a line with such a request can introduce unwanted space into a string.

```
.ds UX \s-1UNIX\s0\u\s-3tm\s0\d \" UNIX trademark
```

Instead the comment should be put on another line or have the comment escape adjacent with the end of the string.

```
.ds UX \s-1UNIX\s0\u\s-3tm\s0\d\" UNIX trademark
```

To produce leading space the string can be started with a double quote. No trailing quote is needed; in fact, any trailing quote is included in your string.

```
.ds sign " Yours in a white wine sauce,
```

Strings are not limited to a single line of text. A string can span several lines by escaping the newlines with a backslash. The resulting string is stored *without* the newlines.

```
.ds foo lots and lots \
of text are on these \
next several lines
```

It is not possible to have real newlines in a string. To put a single double quote character into a string, use two consecutive double quote characters.

The `ds1` request turns off compatibility mode while interpreting a string. To be more precise, a *compatibility save* input token is inserted at the beginning of the string, and a *compatibility restore* input token at the end.

```
.nr xxx 12345
.ds aa The value of xxx is \n[xxx].
.ds1 bb The value of xxx is \n[xxx].
.
.cp 1
.
*(aa
=> warning: number register '[' not defined
=> The value of xxx is 0xxx].
*(bb
=> The value of xxx is 12345.
```

Strings, macros, and diversions (and boxes) share the same name space. Internally, even the same mechanism is used to store them. This has some interesting consequences. For example, it is possible to call a macro with string syntax and vice versa.

```
.de xxx
a funny test.
..
This is *[xxx]
=> This is a funny test.

.ds yyy a funny test
This is
.YYY
=> This is a funny test.
```

In particular, interpolating a string does not hide existing macro arguments. Thus in a macro, a more efficient way of doing

```
.xx \${@}
```

is

```
*[xx]\
```

Note that the latter calling syntax doesn't change the value of `\$0`, which is then inherited from the calling macro.

Diversions and boxes can be also called with string syntax.

Another consequence is that you can copy one-line diversions or boxes to a string.

```
.di xxx
a \fItest\fR
.br
.di
.ds yyy This is *[xxx]\c
*[yyy].
=> This is a test.
```

As the previous example shows, it is possible to store formatted output in strings. The `\c` escape prevents the insertion of an additional blank line in the output.

Copying diversions longer than a single output line produces unexpected results.

```
.di xxx
a funny
.br
test
.br
.di
.ds yyy This is *[xxx]\c
*[yyy].
=> test This is a funny.
```

Usually, it is not predictable whether a diversion contains one or more output lines, so this mechanism should be avoided. With `UNIXtroff`, this was the only solution to strip off a final newline from a diversion. Another disadvantage is that the spaces in

the copied string are already formatted, making them unstretchable. This can cause ugly results.

A clean solution to this problem is available in GNU `troff`, using the requests `chop` to remove the final newline of a diversion, and `unformat` to make the horizontal spaces stretchable again.

```
.box xxx
a funny
.br
test
.br
.box
.chop xxx
.unformat xxx
This is *[xxx].
=> This is a funny test.
```

See [gtroff Internals](#), for more information.

`.as` *name* [*string*]

`.as1` *name* [*string*]

The `as` request is similar to `ds` but appends *string* to the string stored as *name* instead of redefining it. If *name* doesn't exist yet, it is created.

```
.as sign " with shallots, onions and garlic,
```

The `as1` request is similar to `as`, but compatibility mode is switched off while the appended string is interpreted. To be more precise, a *compatibility save* input token is inserted at the beginning of the appended string, and a *compatibility restore* input token at the end.

Rudimentary string manipulation routines are given with the next two requests.

`.substring` *str* *n1* [*n2*]

Replace the string named *str* with the substring defined by the indices *n1* and *n2*. The first character in the string has index 0. If *n2* is omitted, it is implicitly set to the largest valid value (the string length minus one). If the index value *n1* or *n2* is negative, it is counted from the end of the string, going backwards: The last character has index -1, the character before the last character has index -2, etc.

```
.ds xxx abcdefgh
.substring xxx 1 -4
*[xxx]
=> bcde
.substring xxx 2
*[xxx]
=> de
```

`.length` *reg* *str*

Compute the number of characters of *str* and return it in the number register *reg*. If *reg* doesn't exist, it is created. *str* is read in copy mode.

```
.ds xxx abcd\h'3i'efgh
.length yyy *[xxx]
\n[yyy]
=> 14
```

`.rn xx yy`

Rename the request, macro, diversion, or string *xx* to *yy*.

`.rm xx`

Remove the request, macro, diversion, or string *xx*. `gtrouff` treats subsequent invocations as if the object had never been defined.

`.als new old`

Create an alias named *new* for the request, string, macro, or diversion object named *old*. The new name and the old name are exactly equivalent (it is similar to a hard rather than a soft link). If *old* is undefined, `gtrouff` generates a warning of type 'mac' and ignores the request.

To understand how the `als` request works it is probably best to think of two different pools: one pool for objects (macros, strings, etc.), and another one for names. As soon as an object is defined, `gtrouff` adds it to the object pool, adds its name to the name pool, and creates a link between them. When `als` creates an alias, it adds a new name to the name pool that gets linked to the same object as the old name.

Now consider this example.

```
.de foo
..
.
.als bar foo
.
.de bar
. foo
..
.
.bar
=> input stack limit exceeded
```

The definition of macro `bar` replaces the old object this name is linked to. However, the alias to `foo` is still active! In other words, `foo` is still linked to the same object as `bar`, and the result of calling `bar` is an infinite, recursive loop that finally leads to an error.

To undo an alias, simply call `rm` on the aliased name. The object itself is not destroyed until there are no more aliases.

`.chop xx`

Remove (chop) the last character from the macro, string, or diversion named *xx*. This is useful for removing the newline from the end of diversions that are to be interpolated as strings. This command can be used repeatedly; see [gtrouff Internals](#), for details on nodes inserted additionally by `gtrouff`.

See [Identifiers](#), and [Comments](#).

## 5.20. Conditionals and Loops

### 5.20.1. Operators in Conditionals

In `if`, `ie`, and `while` requests, in addition to ordinary [Expressions](#), there are several more operators available:

- e  
o True if the current page is even or odd numbered (respectively).
- n True if the document is being processed in nroff mode (i.e., the `.nroff` command has been issued). See [Troff and Nroff Mode](#).
- t True if the document is being processed in troff mode (i.e., the `.troff` command has been issued). See [Troff and Nroff Mode](#).
- v Always false. This condition is for compatibility with other `troff` versions only (identifying a `-Tversatec` device).

`'xxx' yyy'`

True if the output produced by `xxx` is equal to the output produced by `yyy`. Other characters can be used in place of the single quotes; the same set of delimiters as for the `\D` escape is used (see [Escapes](#)). `gtroff` formats `xxx` and `yyy` in separate environments; after the comparison the resulting data is discarded.

```
.ie "|\"fR|\"fP" \
true
.el \
false
=> true
```

The resulting motions, glyph sizes, and fonts have to match,<sup>22</sup> `--gtroff Internals` and not the individual motion, size, and font requests. In the previous example, `'|'` and `'\fR|\fP'` both result in a roman `|` glyph with the same point size and at the same location on the page, so the strings are equal. If `'.ft l'` had been added before the `'.ie'`, the result would be "false" because (the first) `|` produces an italic `|` rather than a roman one.

To compare strings without processing, surround the data with `\?`.

```
.ie "\?|\?"\?"\fR|\fP\?" \
true
.el \
false
=> false
```

Since data protected with `\?` is read in copy-in mode it is even possible to use incomplete input without causing an error.

```
.ds a \[
.ds b \[
.ie '\?\'*a\?'\'*\b\?' \
true
.el \
false
=> true
```

- r `xxx` True if there is a number register named `xxx`.
- d `xxx` True if there is a string, macro, diversion, or request named `xxx`.

<sup>22</sup> The created output nodes must be identical. See [gtroff Internals](#).

- m** *xxx* True if there is a color named *xxx*.
- c** *g* True if there is a glyph *g* available<sup>23</sup>; *g* is either an ASCII character or a special character (`\N' xxx'`, `\(gg` or `\[ggg]`); the condition is also true if *g* has been defined by the `char` request.
- F** *font* True if a font named *font* exists. *font* is handled as if it was opened with the `ft` request (that is, font translation and styles are applied), without actually mounting it.  
This test doesn't load the complete font but only its header to verify its validity.
- S** *style* True if style *style* has been registered. Font translation is applied.

Note that these operators can't be combined with other operators like ':' or '&'; only a leading '!' (without whitespace between the exclamation mark and the operator) can be used to negate the result.

```
.nr xxx 1
.ie !r xxx \
true
.el \
false
=> false
```

A whitespace after '!' always evaluates to zero (this bizarre behaviour is due to compatibility with UNIX `troff`).

```
.nr xxx 1
.ie ! r xxx \
true
.el \
false
=> r xxx true
```

It is possible to omit the whitespace before the argument to the 'r', 'd', and 'c' operators.

See [Expressions](#).

### 5.20.2. if-else

`gtroff` has if-then-else constructs like other languages, although the formatting can be painful.

`.if` *expr anything*

Evaluate the expression *expr*, and executes *anything* (the remainder of the line) if *expr* evaluates to a value greater than zero (true). *anything* is interpreted as though it was on a line by itself (except that leading spaces are swallowed). See [Operators in Conditionals](#), for more info.

```
.nr xxx 1
.nr yyy 2
.if ((\n[xxx] == 1) & (\n[yyy] == 2)) true
=> true
```

`.nop` *anything*

<sup>23</sup> The name of this conditional operator is a misnomer since it tests names of output glyphs.

Executes *anything*. This is similar to `.if 1`.

```
.ie expr anything
```

```
.el anything
```

Use the `ie` and `el` requests to write an if-then-else. The first request is the ‘if’ part and the latter is the ‘else’ part.

```
.ie n .ls 2 \" double-spacing in nroff
.el .ls 1 \" single-spacing in troff
```

In many cases, an if (or if-else) construct needs to execute more than one request. This can be done using the escapes `\{` (which must start the first line) and `\}` (which must end the last line).

```
.ie t \{\
. ds lq \"
. ds rq \"
.\}
.el \{\
. ds lq ""
. ds rq ""
.\}
```

See [Expressions](#).

### 5.20.3. while

`gtroff` provides a looping construct using the `while` request, which is used much like the `if` (and related) requests.

```
.while expr anything
```

Evaluate the expression *expr*, and repeatedly execute *anything* (the remainder of the line) until *expr* evaluates to 0.

```
.nr a 0 1
.while (\na < 9) \{\
\n+a,
.\}
\n+a
=> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
```

Some remarks.

- The body of a `while` request is treated like the body of a `de` request: `gtroff` temporarily stores it in a macro that is deleted after the loop has been exited. It can considerably slow down a macro if the body of the `while` request (within the macro) is large. Each time the macro is executed, the `while` body is parsed and stored again as a temporary macro.

```
.de xxx
. nr num 10
. while (\n[num] > 0) \{\
. \" many lines of code
. nr num -1
. \}
..
```

The traditional and often better solution (UNIX `troff` doesn't have the `while` request) is to use a recursive macro instead that is parsed only once during its definition.

```
.de yyy
. if (\n[num] > 0) \{\
. \" many lines of code
. nr num -1
. YYY
. \}
..
.
.de xxx
. nr num 10
. YYY
..
```

Note that the number of available recursion levels is set to 1000 (this is a compile-time constant value of `gtroff`).

- The closing brace of a `while` body must end a line.

```
.if 1 \{\
. nr a 0 1
. while (\n[a] < 10) \{\
. nop \n+[a]
.\}\}
=> unbalanced \{ \}
```

`.break`

Break out of a `while` loop. Be sure not to confuse this with the `br` request (causing a line break).

`.continue`

Finish the current iteration of a `while` loop, immediately restarting the next iteration.

See [Expressions](#).

## 5.21. Writing Macros

A *macro* is a collection of text and embedded commands that can be invoked multiple times. Use macros to define common operations. See [Strings](#), for a (limited) alternative syntax to call macros.

Although the following requests can be used to create macros, simply using an undefined macro will cause it to be defined as empty. See [Identifiers](#).

```
.de name [end]
.de1 name [end]
.dei name [end]
.dei1 name [end]
```

Define a new macro named *name*. `gtroff` copies subsequent lines (starting with the next one) into an internal buffer until it encounters the line `..'` (two dots). If the optional second argument to `de` is present it is used as the macro closure request instead of `..'`.

There can be whitespace after the first dot in the line containing the ending token (either `'.` or macro `'end'`). Don't insert a tab character immediately after the `'.`, otherwise it isn't recognized as the end-of-macro symbol. While it is possible to define and call a macro `'.` with

```
.de .
. tm foo
..
.
.. \" This calls macro \.'
```

you can't use this as the end-of-macro macro: during a macro definition, `'.` is never handled as a call to `'.`, even if you say `'de foo .' explicitly.`

Here a small example macro called `'P'` that causes a break and inserts some vertical space. It could be used to separate paragraphs.

```
.de P
. br
. sp .8v
..
```

The following example defines a macro within another. Remember that expansion must be protected twice; once for reading the macro and once for executing.

```
\# a dummy macro to avoid a warning
.de end
..
.
.de foo
. de bar end
. nop \f[B]Hello \\$1!\f[]
. end
..
.
.foo
.bar Joe
=> Hello Joe!
```

Since `\f` has no expansion, it isn't necessary to protect its backslash. Had we defined another macro within `bar` that takes a parameter, eight backslashes would be necessary before `'$1'`.

The `de1` request turns off compatibility mode while executing the macro. On entry, the current compatibility mode is saved and restored at exit.

```
.nr xxx 12345
.
.de aa
The value of xxx is \n[xxx].
..
.de1 bb
The value of xxx is \n[xxx].
..
.
```

```
.cp 1
.
.aa
=> warning: number register '[' not defined
=> The value of xxx is 0xxx].
.bb
=> The value of xxx is 12345.
```

The `dei` request defines a macro indirectly. That is, it expands strings whose names are *name* or *end* before performing the append.

This:

```
.ds xx aa
.ds yy bb
.dei xx yy
```

is equivalent to:

```
.de aa bb
```

The `dei1` request is similar to `dei` but with compatibility mode switched off during execution of the defined macro.

If compatibility mode is on, `de` (and `dei`) behave similar to `de1` (and `dei1`): A ‘compatibility save’ token is inserted at the beginning, and a ‘compatibility restore’ token at the end, with compatibility mode switched on during execution. See [gtroff Internals](#), for more information on switching compatibility mode on and off in a single document.

Using `trace.tmac`, you can trace calls to `de` and `de1`.

Note that macro identifiers are shared with identifiers for strings and diversions.

See [the description of the `als` request](#), for possible pitfalls if redefining a macro that has been aliased.

```
.am name [end]
.am1 name [end]
.ami name [end]
.ami1 name [end]
```

Works similarly to `de` except it appends onto the macro named *name*. So, to make the previously defined ‘P’ macro actually do indented instead of block paragraphs, add the necessary code to the existing macro like this:

```
.am P
.ti +5n
..
```

The `am1` request turns off compatibility mode while executing the appended macro piece. To be more precise, a *compatibility save* input token is inserted at the beginning of the appended code, and a *compatibility restore* input token at the end.

The `ami` request appends indirectly, meaning that `gtroff` expands strings whose names are *name* or *end* before performing the append.

The `ami1` request is similar to `ami` but compatibility mode is switched off during execution of the defined macro.

Using `trace.tmac`, you can trace calls to `am` and `am1`.

See [Strings](#), for the `als` and `rn` request to create an alias and rename a macro, respectively.

The `de`, `am`, `di`, `da`, `ds`, and `as` requests (together with their variants) only create a new object if the name of the macro, diversion or string is currently undefined or if it is defined to be a request; normally they modify the value of an existing object.

`.return [anything]`

Exit a macro, immediately returning to the caller.

If called with an argument, exit twice, namely the current macro and the macro one level higher. This is used to define a wrapper macro for `return` in `trace.tmac`.

### 5.21.1. Copy-in Mode

When `gtrouff` reads in the text for a macro, string, or diversion, it copies the text (including request lines, but excluding escapes) into an internal buffer. Escapes are converted into an internal form, except for `\n`, `\$`, `\*`, `\` and `\RET`, which are evaluated and inserted into the text where the escape was located. This is known as *copy-in* mode or *copy* mode.

What this means is that you can specify when these escapes are to be evaluated (either at copy-in time or at the time of use) by insulating the escapes with an extra backslash. Compare this to the `\def` and `\edef` commands in `TEX`.

The following example prints the numbers 20 and 10:

```
.nr x 20
.de y
.nr x 10
&\nx
&\nx
..
.y
```

### 5.21.2. Parameters

The arguments to a macro or string can be examined using a variety of escapes.

`\n[. $]`

The number of arguments passed to a macro or string. This is a read-only number register.

Note that the `shift` request can change its value.

Any individual argument can be retrieved with one of the following escapes:

`\$n`

`\$(nn`

`\$[nnn]`

Retrieve the *n*th, *nn*th or *nnn*th argument. As usual, the first form only accepts a single number (larger than zero), the second a two-digit number (larger or equal to 10), and the third any positive integer value (larger than zero). Macros and strings can have an unlimited number of arguments. Note that due to copy-in mode, use two backslashes on these in actual use to prevent interpolation until the macro is actually invoked.

`.shift [n]`

Shift the arguments 1 position, or as many positions as specified by its argument. After executing this request, argument *i* becomes argument *i-@var{n}*; arguments 1 to *n* are no longer available. Shifting by negative amounts is currently undefined.

The register `.$` is adjusted accordingly.

`\$*``\$@`

In some cases it is convenient to use all of the arguments at once (for example, to pass the arguments along to another macro). The `\$*` escape concatenates all the arguments separated by spaces. A similar escape is `\$@`, which concatenates all the arguments with each surrounded by double quotes, and separated by spaces. If not in compatibility mode, the input level of double quotes is preserved (see [Request and Macro Arguments](#)).

`\$^`

Handle the parameters of a macro as if they were an argument to the `ds` or similar requests.

```
.de foo
. tm $1=``\$1'
. tm $2=``\$2'
. tm $*=``\$*'
. tm $@=``\$@'
. tm $^=``\$^'
..
.foo " This is a "test"
=> $1=` This is a '
=> $2=`test"'
=> $*=` This is a test"'
=> $@=`` This is a " "test"'
=> $^=`` This is a "test"'
```

This escape is useful mainly for macro packages like `trace.tmac`, which redefines some requests and macros for debugging purposes.

`\$0`

The name used to invoke the current macro. The `als` request can make a macro have more than one name.

If a macro is called as a string (within another macro), the value of `\$0` isn't changed.

```
.de foo
. tm \$0
..
.als foo bar
.
.de aaa
. foo
..
.de bbb
. bar
```

```

..
.de ccc
\[foo]\
..
.de ddd
\[bar]\
..
.
.aaa
=> foo
.bbb
=> bar
.ccc
=> ccc
.ddd
=> ddd

```

See [Request and Macro Arguments](#).

## 5.22. Page Motions

See [Manipulating Spacing](#), for a discussion of the main request for vertical motion, `sp`.

`.mk` [*reg*]

`.rt` [*dist*]

The request `mk` can be used to mark a location on a page, for movement to later. This request takes a register name as an argument in which to store the current page location. With no argument it stores the location in an internal register. The results of this can be used later by the `rt` or the `sp` request (or the `\v` escape).

The `rt` request returns *upwards* to the location marked with the last `mk` request. If used with an argument, return to a position which distance from the top of the page is *dist* (no previous call to `mk` is necessary in this case). Default scaling indicator is 'v'.

Here a primitive solution for a two-column macro.

```

.nr column-length 1.5i
.nr column-gap 4m
.nr bottom-margin 1m
.
.de 2c
. br
. mk
. ll \n[column-length]u
. wh -\n[bottom-margin]u 2c-trap
. nr right-side 0
..
.
.de 2c-trap
. ie \n[right-side] \{\
. nr right-side 0

```

```

. po -(\n[column-length]u + \n[column-gap]u)
. \" remove trap
. wh -\n[bottom-margin]u
. \}
. el \{\
. \" switch to right side
. nr right-side 1
. po +(\n[column-length]u + \n[column-gap]u)
. rt
. \}
..
.
.pl 1.5i
.ll 4i
This is a small test that shows how the
rt request works in combination with mk.

```

```

.2c
Starting here, text is typeset in two columns.
Note that this implementation isn't robust
and thus not suited for a real two-column
macro.

```

**Result:**

```

This is a small test that shows how the
rt request works in combination with mk.

```

```

Starting here, isn't robust
text is typeset and thus not
in two columns. suited for a
Note that this real two-column
implementation macro.

```

The following escapes give fine control of movements about the page.

**\v'e'**

Move vertically, usually from the current location on the page (if no absolute position operator `'` is used). The argument `e` specifies the distance to move; positive is downwards and negative upwards. The default scaling indicator for this escape is `'v'`. Beware, however, that `gtroff` continues text processing at the point where the motion ends, so you should always balance motions to avoid interference with text processing.

`\v` doesn't trigger a trap. This can be quite useful; for example, consider a page bottom trap macro that prints a marker in the margin to indicate continuation of a footnote or something similar.

There are some special-case escapes for vertical motion.

**\r**

Move upwards 1 v.

`\u`

Move upwards .5v.

`\d`

Move down .5v.

`\h'e'`

Move horizontally, usually from the current location (if no absolute position operator '[' is used). The expression *e* indicates how far to move: positive is rightwards and negative leftwards. The default scaling indicator for this escape is 'm'.

This horizontal space is not discarded at the end of a line. To insert discardable space of a certain length use the `ss` request.

There are a number of special-case escapes for horizontal motion.

`\SP`

An unbreakable and unpaddable (i.e. not expanded during filling) space. (Note: This is a backslash followed by a space.)

`\~`

An unbreakable space that stretches like a normal inter-word space when a line is adjusted.

`\|`

A 1/6 th em space. Ignored for TTY output devices (rounded to zero).

However, if there is a glyph defined in the current font file with name `\|` (note the leading backslash), the width of this glyph is used instead (even for TTYs).

`\^`

A 1/12 th em space. Ignored for TTY output devices (rounded to zero).

However, if there is a glyph defined in the current font file with name `\^` (note the leading backslash), the width of this glyph is used instead (even for TTYs).

`\0`

A space the size of a digit.

The following string sets the T<sub>E</sub>X logo:

```
.ds TeX T\h'-.1667m'\v'.224m'E\v'-.224m'\h'-.125m'X
```

`\w'text'``\n[st]``\n[sb]``\n[rst]``\n[rsb]``\n[ct]``\n[ssc]``\n[skw]`

Return the width of the specified *text* in basic units. This allows horizontal movement based on the width of some arbitrary text (e.g. given as an argument to a macro).

The length of the string 'abc' is `\w'abc'u`.

=> The length of the string 'abc' is 72u.

Font changes may occur in *text*, which don't affect current settings.

After use, `\w` sets several registers:

|                  |                                                                                                                                                                                                                                                                                                                                                          |   |                                                  |   |                         |   |                          |   |                                                    |
|------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---|--------------------------------------------------|---|-------------------------|---|--------------------------|---|----------------------------------------------------|
| <code>st</code>  |                                                                                                                                                                                                                                                                                                                                                          |   |                                                  |   |                         |   |                          |   |                                                    |
| <code>sb</code>  | The highest and lowest point of the baseline, respectively, in <i>text</i> .                                                                                                                                                                                                                                                                             |   |                                                  |   |                         |   |                          |   |                                                    |
| <code>rst</code> |                                                                                                                                                                                                                                                                                                                                                          |   |                                                  |   |                         |   |                          |   |                                                    |
| <code>rsb</code> | Like the <code>st</code> and <code>sb</code> registers, but takes account of the heights and depths of glyphs. In other words, this gives the highest and lowest point of <i>text</i> . Values below the baseline are negative.                                                                                                                          |   |                                                  |   |                         |   |                          |   |                                                    |
| <code>ct</code>  | Defines the kinds of glyphs occurring in <i>text</i> : <table> <tr> <td>0</td> <td>only short glyphs, no descenders or tall glyphs.</td> </tr> <tr> <td>1</td> <td>at least one descender.</td> </tr> <tr> <td>2</td> <td>at least one tall glyph.</td> </tr> <tr> <td>3</td> <td>at least one each of a descender and a tall glyph.</td> </tr> </table> | 0 | only short glyphs, no descenders or tall glyphs. | 1 | at least one descender. | 2 | at least one tall glyph. | 3 | at least one each of a descender and a tall glyph. |
| 0                | only short glyphs, no descenders or tall glyphs.                                                                                                                                                                                                                                                                                                         |   |                                                  |   |                         |   |                          |   |                                                    |
| 1                | at least one descender.                                                                                                                                                                                                                                                                                                                                  |   |                                                  |   |                         |   |                          |   |                                                    |
| 2                | at least one tall glyph.                                                                                                                                                                                                                                                                                                                                 |   |                                                  |   |                         |   |                          |   |                                                    |
| 3                | at least one each of a descender and a tall glyph.                                                                                                                                                                                                                                                                                                       |   |                                                  |   |                         |   |                          |   |                                                    |
| <code>ssc</code> | The amount of horizontal space (possibly negative) that should be added to the last glyph before a subscript.                                                                                                                                                                                                                                            |   |                                                  |   |                         |   |                          |   |                                                    |
| <code>skw</code> | How far to right of the center of the last glyph in the <code>\w</code> argument, the center of an accent from a roman font should be placed over that glyph.                                                                                                                                                                                            |   |                                                  |   |                         |   |                          |   |                                                    |

`\kp`

`\k(ps`

`\k[position]`

Store the current horizontal position in the *input* line in number register with name *position* (one-character name *p*, two-character name *ps*). Use this, for example, to return to the beginning of a string for highlighting or other decoration.

`\n[hp]`

The current horizontal position at the input line.

`\n[.k]`

A read-only number register containing the current horizontal output position (relative to the current indentation).

`\o'abc'`

Overstrike glyphs *a*, *b*, *c*, ...; the glyphs are centered, and the resulting spacing is the largest width of the affected glyphs.

`\zg`

Print glyph *g* with zero width, i.e., without spacing. Use this to overstrike glyphs left-aligned.

`\z'anything'`

Print *anything*, then restore the horizontal and vertical position. The argument may not contain tabs or leaders.

The following is an example of a strike-through macro:

```
.de ST
.nr ww \w' \$1'
\Z@\v' -.25m' \l' \n[ww] u' @\$1
```

```

..
.
This is
.ST "a test"
an actual emergency!

```

### 5.23. Drawing Requests

`gtroff` provides a number of ways to draw lines and other figures on the page. Used in combination with the page motion commands (see [Page Motions](#), for more info), a wide variety of figures can be drawn. However, for complex drawings these operations can be quite cumbersome, and it may be wise to use graphic preprocessors like `gpic` or `ggrn`. See [gpic](#), and [ggrn](#), for more information.

All drawing is done via escapes.

```

\l'l
\l'lg'

```

Draw a line horizontally. `l` is the length of the line to be drawn. If it is positive, start the line at the current location and draw to the right; its end point is the new current location. Negative values are handled differently: The line starts at the current location and draws to the left, but the current location doesn't move.

`l` can also be specified absolutely (i.e. with a leading '|'), which draws back to the beginning of the input line. Default scaling indicator is 'm'.

The optional second parameter `g` is a glyph to draw the line with. If this second argument is not specified, `gtroff` uses the underscore glyph, `\[ru]`.

To separate the two arguments (to prevent `gtroff` from interpreting a drawing glyph as a scaling indicator if the glyph is represented by a single character) use `\&`.

Here a small useful example:

```

.de box
\[br]\$*\[br]\l'l|0\[rn]' \l'l|0\[ul]'
..

```

Note that this works by outputting a box rule (a vertical line), then the text given as an argument and then another box rule. Finally, the line drawing escapes both draw from the current location to the beginning of the *input* line -- this works because the line length is negative, not moving the current point.

```

\L'l
\L'lg'

```

Draw vertical lines. Its parameters are similar to the `\l` escape, except that the default scaling indicator is 'v'. The movement is downwards for positive values, and upwards for negative values. The default glyph is the box rule glyph, `\[br]`. As with the vertical motion escapes, text processing blindly continues where the line ends.

```
This is a \L'3v'test.
```

Here is the result, produced with `grotty`.

```
This is a
 |
 |
```

|test.

`\D'command arg ...'`

The `\D` escape provides a variety of drawing functions. Note that on character devices, only vertical and horizontal lines are supported within `grotty`; other devices may only support a subset of the available drawing functions.

The default scaling indicator for all subcommands of `\D` is 'm' for horizontal distances and 'v' for vertical ones. Exceptions are `\D'f ...'` and `\D't ...'`, which use `u` as the default, and `\D'Fx ...'`, which arguments are treated similar to the `def-color` request.

`\D'l dx dy'`

Draw a line from the current location to the relative point specified by  $(dx,dy)$ , where positive values mean right and down, respectively. The end point of the line is the new current location.

The following example is a macro for creating a box around a text string; for simplicity, the box margin is taken as a fixed value, 0.2 m.

```
.de BOX
. nr @wd \w'\$1'
\h'.2m'\
\h'-.2m'\v'(.2m - \n[rsb]u)'\
\D'l 0 -(\n[rst]u - \n[rsb]u + .4m)'\
\D'l (\n[@wd]u + .4m) 0'\
\D'l 0 (\n[rst]u - \n[rsb]u + .4m)'\
\D'l -(\n[@wd]u + .4m) 0'\
\h'.2m'\v'-.2m - \n[rsb]u)'\
\$1\
\h'.2m'
..
```

First, the width of the string is stored in register `@wd`. Then, four lines are drawn to form a box, properly offset by the box margin. The registers `rst` and `rsb` are set by the `\w` escape, containing the largest height and depth of the whole string.

`\D'c d'` Draw a circle with a diameter of  $d$  with the leftmost point at the current position. After drawing, the current location is positioned at the rightmost point of the circle.

`\D'C d'` Draw a solid circle with the same parameters and behaviour as an outlined circle. No outline is drawn.

`\D'e x y'`

Draw an ellipse with a horizontal diameter of  $x$  and a vertical diameter of  $y$  with the leftmost point at the current position. After drawing, the current location is positioned at the rightmost point of the ellipse.

`\D'E x y'`

Draw a solid ellipse with the same parameters and behaviour as an outlined ellipse. No outline is drawn.

`\D'a dx1 dy1 dx2 dy2'`

Draw an arc clockwise from the current location through the two specified relative locations  $(dx1,dy1)$  and  $(dx2,dy2)$ . The coordinates of the first point are relative to the current position, and the coordinates of the second point are relative to the first point. After drawing, the current position is moved to the final point of the arc.

`\D'~ dx1 dy1 dx2 dy2 ...'`

Draw a spline from the current location to the relative point  $(dx1,dy1)$  and then to  $(dx2,dy2)$ , and so on. The current position is moved to the terminal point of the drawn curve.

`\D'f n'` Set the shade of gray to be used for filling solid objects to  $n$ ;  $n$  must be an integer between 0 and 1000, where 0 corresponds solid white and 1000 to solid black, and values in between correspond to intermediate shades of gray. This applies only to solid circles, solid ellipses, and solid polygons. By default, a level of 1000 is used.

Despite of being silly, the current point is moved horizontally to the right by  $n$ .

Don't use this command! It has the serious drawback that it is always rounded to the next integer multiple of the horizontal resolution (the value of the `hor` keyword in the `DESC` file). Use `\M` (see [Colors](#)) or `\D'Fg ...'` instead.

`\D'p dx1 dy1 dx2 dy2 ...'`

Draw a polygon from the current location to the relative position  $(dx1,dy1)$  and then to  $(dx2,dy2)$  and so on. When the specified data points are exhausted, a line is drawn back to the starting point. The current position is changed by adding the sum of all arguments with odd index to the actual horizontal position and the even ones to the vertical position.

`\D'P dx1 dy1 dx2 dy2 ...'`

Draw a solid polygon with the same parameters and behaviour as an outlined polygon. No outline is drawn.

Here a better variant of the `box` macro to fill the box with some color. Note that the box must be drawn before the text since colors in `gtroff` are not transparent; the filled polygon would hide the text completely.

```
.de BOX
. nr @wd \w'\$1'
\h'.2m'\
\h'-.2m'\v'(.2m - \n[rsb]u)'\
\M[lightcyan]\
\D'P 0 -(\n[rst]u - \n[rsb]u + .4m) \
 (\n[@wd]u + .4m) 0 \
 0 (\n[rst]u - \n[rsb]u + .4m) \
 -(\n[@wd]u + .4m) 0'\
\h'.2m'\v'-(.2m - \n[rsb]u)'\
```

```

\M[]\
\$1\
\h' .2m'
..

```

If you want a filled polygon that has exactly the same size as an unfilled one, you must draw both an unfilled and a filled polygon. A filled polygon is always smaller than an unfilled one because the latter uses straight lines with a given line thickness to connect the polygon's corners, while the former simply fills the area defined by the coordinates.

```

\h'1i'\v'1i'\
\# increase line thickness
\Z'\D't 5p"\
\# draw unfilled polygon
\Z'\D'p 3 3 -6 0"\
\# draw filled polygon
\Z'\D'P 3 3 -6 0"

```

`\D't n'` Set the current line thickness to  $n$  machine units. A value of zero selects the smallest available line thickness. A negative value makes the line thickness proportional to the current point size (this is the default behaviour of AT&T `troff`).

Despite of being silly, the current point is moved horizontally to the right by  $n$ .

`\D'Fscheme color_components'`

Change current fill color. *scheme* is a single letter denoting the color scheme: 'r' (rgb), 'c' (cmy), 'k' (cmyk), 'g' (gray), or 'd' (default color). The color components use exactly the same syntax as in the `def-color` request (see [Colors](#)); the command `\D'Fd'` doesn't take an argument.

No position changing!

Examples:

```

\D'Fg .3' \" same gray as \D'f 700'
\D'Fr #0000ff' \" blue

```

See [Graphics Commands](#). `\b'string'`

*Pile* a sequence of glyphs vertically, and center it vertically on the current line. Use it to build large brackets and braces.

Here an example how to create a large opening brace:

```

\b'\[lt]\[bv]\[lk]\[bv]\[lb]'

```

The first glyph is on the top, the last glyph in *string* is at the bottom. Note that `gtroff` separates the glyphs vertically by 1 m, and the whole object is centered 0.5 m above the current baseline; the largest glyph width is used as the width for the whole object. This rather inflexible positioning algorithm doesn't work with `-Tdvi` since the bracket pieces vary in height for this device. Instead, use the `eqn` pre-processor.

See [Manipulating Spacing](#), how to adjust the vertical spacing with the `\x` escape.

## 5.24. Traps

*Traps* are locations that, when reached, call a specified macro. These traps can occur at a given location on the page, at a given location in the current diversion, at a blank line, after a certain number of input lines, or at the end of input.

Setting a trap is also called *planting*. It is also said that a trap is *sprung* if the associated macro is executed.

### 5.24.1. Page Location Traps

*Page location traps* perform an action when `gtruff` reaches or passes a certain vertical location on the page. Page location traps have a variety of purposes, including:

- setting headers and footers
- setting body text in multiple columns
- setting footnotes

`.vpt` *flag*

`\n[.vpt]`

Enable vertical position traps if *flag* is non-zero, or disables them otherwise. Vertical position traps are traps set by the `wh` or `dt` requests. Traps set by the `it` request are not vertical position traps. The parameter that controls whether vertical position traps are enabled is global. Initially vertical position traps are enabled. The current setting of this is available in the `.vpt` read-only number register.

Note that a page can't be ejected if `vpt` is set to zero.

`.wh` *dist* [*macro*]

Set a page location trap. Non-negative values for *dist* set the trap relative to the top of the page; negative values set the trap relative to the bottom of the page. Default scaling indicator is 'v'; values of *dist* are always rounded to be multiples of the vertical resolution (as given in register `.v`).

*macro* is the name of the macro to execute when the trap is sprung. If *macro* is missing, remove the first trap (if any) at *dist*.

The following is a simple example of how many macro packages set headers and footers.

```
.de hd \" Page header
' sp .5i
. tl 'Title"date'
' sp .3i
..
.
.de fo \" Page footer
' sp 1v
. tl ""%"
' bp
..
```

```

.
.wh 0 hd \" trap at top of the page
.wh -1i fo \" trap one inch from bottom

```

A trap at or below the bottom of the page is ignored; it can be made active by either moving it up or increasing the page length so that the trap is on the page.

Negative trap values always use the *current* page length; they are not converted to an absolute vertical position:

```

.pl 5i
.wh -1i xx
.ptr
=> xx -240
.pl 100i
.ptr
=> xx -240

```

It is possible to have more than one trap at the same location; to do so, the traps must be defined at different locations, then moved together with the `ch` request; otherwise the second trap would replace the first one. Earlier defined traps hide later defined traps if moved to the same position (the many empty lines caused by the `bp` request are omitted in the following example):

```

.de a
. nop a
..
.de b
. nop b
..
.de c
. nop c
..
.
.wh 1i a
.wh 2i b
.wh 3i c
.bp
=> a b c

.ch b 1i
.ch c 1i
.bp
=> a

.ch a 0.5i
.bp
=> a b

```

`\n[.t]`

A read-only number register holding the distance to the next trap.

If there are no traps between the current position and the bottom of the page, it contains the distance to the page bottom. In a diversion, the distance to the page bottom is infinite (the returned value is the biggest integer that can be represented in

`gtruff`) if there are no diversion traps.

`.ch` *macro* [*dist*]

Change the location of a trap. The first argument is the name of the macro to be invoked at the trap, and the second argument is the new location for the trap (note that the parameters are specified in opposite order as in the `wh` request). This is useful for building up footnotes in a diversion to allow more space at the bottom of the page for them.

Default scaling indicator for *dist* is 'v'. If *dist* is missing, the trap is removed.

`\n[.ne]`

The read-only number register `.ne` contains the amount of space that was needed in the last `ne` request that caused a trap to be sprung. Useful in conjunction with the `.trunc` register. See [Page Control](#), for more information.

Since the `.ne` register is only set by traps it doesn't make much sense to use it outside of trap macros.

`\n[.trunc]`

A read-only register containing the amount of vertical space truncated from an `sp` request by the most recently sprung vertical position 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, it represents the difference of what the vertical position would have been but for the trap, and what the vertical position actually is.

Since the `.trunc` register is only set by traps it doesn't make much sense to use it outside of trap macros.

`\n[.pe]`

A read-only register that is set to 1 while a page is ejected with the `bp` request (or by the end of input).

Outside of traps this register is always zero. In the following example, only the second call to `x` is caused by `bp`.

```
.de x
\&.pe=\n[.pe]
.br
..
.wh 1v x
.wh 4v x
A line.
.br
Another line.
.br
=> A line.
.pe=0
Another line.

.pe=1
```

An important fact to consider while designing macros is that diversions and traps do not interact normally. For example, if a trap invokes a header macro (while outputting a diversion) that tries to change the font on the current page, the effect is not visible before the

diversion has completely been printed (except for input protected with \! or \?) since the data in the diversion is already formatted. In most cases, this is not the expected behaviour.

### 5.24.2. Diversion Traps

`.dt` [*dist macro*]

Set a trap *within* a diversion. *dist* is the location of the trap (identical to the `wh` request; default scaling indicator is 'v') and *macro* is the name of the macro to be invoked. If called without arguments, the diversion trap is removed.

Note that there exists only a single diversion trap.

The number register `.t` still works within diversions. See [Diversions](#), for more information.

### 5.24.3. Input Line Traps

`.it` *n macro*

`.itc` *n macro*

Set an input line trap. *n* is the number of lines of input that may be read before springing the trap, *macro* is the macro to be invoked. Request lines are not counted as input lines.

For example, one possible use is to have a macro that prints the next *n* lines in a bold font.

```
.de B
. it \ $1 B-end
. ft B
..
.
.de B-end
. ft R
..
```

The `itc` request is identical except that an interrupted text line (ending with `\c`) is not counted as a separate line.

Both requests are associated with the current environment (see [Environments](#)); switching to another environment disables the current input trap, and going back re-activates it, restoring the number of already processed lines.

### 5.24.4. Blank Line Traps

`.blm` *macro*

Set a blank line trap. `gtroff` executes *macro* when it encounters a blank line in the input file.

### 5.24.5. Leading Spaces Traps

`.lsm` *macro*

`\n[lsn]`

`\n[lss]`

Set a leading spaces trap. `gtroff` executes *macro* when it encounters leading spaces in an input line; the implicit line break that normally happens in this case is suppressed. A line consisting of spaces only, however, is treated as an empty line, possibly subject to an empty line macro set with the `blm` request.

Leading spaces are removed from the input line before calling the leading spaces macro. The number of removed spaces is stored in register `lsn`; the horizontal space that would be emitted if there was no leading space macro is stored in register `lss`. Note that `lsn` and `lss` are available even if no leading space macro has been set.

The first thing a leading space macro sees is a token. However, some escapes like `\f` or `\m` are handled on the fly (see [gtroff Internals](#), for a complete list) without creating a token at all. Consider that a line starts with two spaces followed by `\fIfoo`. While skipping the spaces `\fI` is handled too so that `gtroff`'s current font is properly set to 'I', but the leading space macro only sees `foo`, without the preceding `\fI`. If the macro should see the font escape you have to 'protect' it with something that creates a token, for example with `&\fIfoo`.

#### 5.24.6. End-of-input Traps

`.em` *macro*

Set a trap at the end of input. *macro* is executed after the last line of the input file has been processed.

For example, if the document had to have a section at the bottom of the last page for someone to approve it, the `em` request could be used.

```
.de approval
 \c
 . ne 3v
 . sp (\n[.t]u - 3v)
 . in +4i
 . lc _
 . br
Approved:\t\a
 . sp
Date:\t\t\a
 ..
 .
.em approval
```

The `\c` in the above example needs explanation. For historical reasons (and for compatibility with AT&T `troff`), the end macro exits as soon as it causes a page break and no remaining data is in the partially collected line.

Let us assume that there is no `\c` in the above `approval` macro, and that the page is full and has been ended with, say, a `br` request. The `ne` request now causes the start of a new page, which in turn makes `troff` exit immediately for the reasons just described. In most situations this is not intended.

To always force processing the whole end macro independently of this behaviour it is thus advisable to insert something that starts an empty partially filled line (`\c`) whenever there is a chance that a page break can happen. In the above example, the call

of the `ne` request assures that the remaining code stays on the same page, so we have to insert `\c` only once.

The next example shows how to append three lines, then starting a new page unconditionally. Since `'ne 1'` doesn't give the desired effect -- there is always one line available or we are already at the beginning of the next page -- we temporarily increase the page length by one line so that we can use `'ne 2'`.

```
.de EM
.pl +1v
\c
.ne 2
line one
.br
\c
.ne 2
line two
.br
\c
.ne 2
line three
.br
.pl -1v
\c
'bp
..
.em EM
```

Note that this specific feature affects only the first potential page break caused by the end macro; further page breaks emitted by the end macro are handled normally.

Another possible use of the `em` request is to make `gtroff` emit a single large page instead of multiple pages. For example, one may want to produce a long plain-text file for reading on-screen. The idea is to set the page length at the beginning of the document to a very large value to hold all the text, and automatically adjust it to the exact height of the document after the text has been output.

```
.de adjust-page-length
. br
. pl \n[nl]u \" \n[nl] holds the current page length
..
.
.de single-page-mode
. pl 99999
. em adjust-page-length
..
.
.\" activate the above code
.single-page-mode
```

Since only one end-of-input trap does exist and other macro packages may already use it, care must be taken not to break the mechanism. A simple solution would be to append the above macro to the macro package's end-of-input macro using the

.am request.

## 5.25. Diversions

In `gtroff` it is possible to *divert* text into a named storage area. Due to the similarity to defining macros it is sometimes said to be stored in a macro. This is used for saving text for output at a later time, which is useful for keeping blocks of text on the same page, footnotes, tables of contents, and indices.

For orthogonality it is said that `gtroff` is in the *top-level diversion* if no diversion is active (i.e., the data is diverted to the output device).

Although the following requests can be used to create diversions, simply using an undefined diversion will cause it to be defined as empty. See [Identifiers](#).

`.di macro`

`.da macro`

Begin a diversion. Like the `de` request, it takes an argument of a macro name to divert subsequent text into. The `da` macro appends to an existing diversion.

`di` or `da` without an argument ends the diversion.

The current partially filled line is included into the diversion. See the `box` request below for an example. Note that switching to another (empty) environment (with the `ev` request) avoids the inclusion of the current partially filled line.

`.box macro`

`.boxa macro`

Begin (or append to) a diversion like the `di` and `da` requests. The difference is that `box` and `boxa` do not include a partially filled line in the diversion.

Compare this:

```
Before the box.
.box xxx
In the box.
.br
.box
After the box.
.br
=> Before the box. After the box.
.xxx
=> In the box.
```

with this:

```
Before the diversion.
.di yyy
In the diversion.
.br
.di
After the diversion.
.br
=> After the diversion.
.YYY
=> Before the diversion. In the diversion.
```

`box` or `boxa` without an argument ends the diversion.

`\n[.z]`

`\n[.d]`

Diversions may be nested. The read-only number register `.z` contains the name of the current diversion (this is a string-valued register). The read-only number register `.d` contains the current vertical place in the diversion. If not in a diversion it is the same as register `nl`.

`\n[.h]`

The *high-water mark* on the current page or in the current diversion. It corresponds to the text baseline of the lowest line on the page. This is a read-only register.

```
.tm .h==\n[.h], nl==\n[nl]
=> .h==0, nl==-1
This is a test.
.br
.sp 2
.tm .h==\n[.h], nl==\n[nl]
=> .h==40, nl==120
```

As can be seen in the previous example, empty lines are not considered in the return value of the `.h` register.

`\n[dn]`

`\n[d1]`

After completing a diversion, the read-write number registers `dn` and `d1` contain the vertical and horizontal size of the diversion. Note that only the just processed lines are counted: For the computation of `dn` and `d1`, the requests `da` and `boxa` are handled as if `di` and `box` had been used -- lines that have been already stored in a macro are not taken into account.

```
.\" Center text both horizontally & vertically
.
.\" Enclose macro definitions in .eo and .ec
.\" to avoid the doubling of the backslash
.eo
.\" macro .(c starts centering mode
.de (c
. br
. ev (c
. evc 0
. in 0
. nf
. di @c
..
.\" macro .)c terminates centering mode
.de)c
. br
. ev
. di
. nr @s (((\n[.t]u - \n[dn]u) / 2u) - 1v)
. sp \n[@s]u
```

```

. ce 1000
. @c
. ce 0
. sp \n[@s]u
. br
. fi
. rr @s
. rm @s
. rm @c
..
.\ " End of macro definitions, restore escape mechanism
.ec

```

\!

\?*anything*?

Prevent requests, macros, and escapes from being interpreted when read into a diversion. Both escapes take the given text and *transparently* embed it into the diversion. This is useful for macros that shouldn't be invoked until the diverted text is actually output.

The \! escape transparently embeds text up to and including the end of the line. The \? escape transparently embeds text until the next occurrence of the \? escape. Example:

```
\?anything?
```

*anything* may not contain newlines; use \! to embed newlines in a diversion. The escape sequence \? is also recognized in copy mode and turned into a single internal code; it is this code that terminates *anything*. Thus the following example prints 4.

```

.nr x 1
.nf
.di d
\?\?\?\?\?\?\nx\?\?\?
.di
.nr x 2
.di e
.d
.di
.nr x 3
.di f
.e
.di
.nr x 4
.f

```

Both escapes read the data in copy mode.

If \! is used in the top-level diversion, its argument is directly embedded into the `gtroff` intermediate output. This can be used for example to control a postprocessor that processes the data before it is sent to the device driver.

The \? escape used in the top-level diversion produces no output at all; its argument

is simply ignored.

#### `.output string`

Emit *string* directly to the `gtroff` intermediate output (subject to copy mode interpretation); this is similar to `\!` used at the top level. An initial double quote in *string* is stripped off to allow initial blanks.

This request can't be used before the first page has started -- if you get an error, simply insert `.br` before the `output` request.

Without argument, `output` is ignored.

Use with caution! It is normally only needed for mark-up used by a postprocessor that does something with the output before sending it to the output device, filtering out *string* again.

#### `.asciify div`

*Unformat* the diversion specified by *div* in such a way that ASCII characters, characters translated with the `trin` request, space characters, and some escape sequences that were formatted and diverted are treated like ordinary input characters when the diversion is reread. It can be also used for gross hacks; for example, the following sets register `n` to 1.

```
.tr @.
.di x
@nr n 1
.br
.di
.tr @@
.asciify x
.x
```

Note that `asciify` cannot return all items in a diversion back to their source equivalent, nodes such as `\N[...]` will still remain as nodes, so the result cannot be guaranteed to be a pure string.

See [Copy-in Mode](#).

#### `.unformat div`

Like `asciify`, `unformat` the specified diversion. However, `unformat` only unformats spaces and tabs between words. Unformatted tabs are treated as input tokens, and spaces are stretchable again.

The vertical size of lines is not preserved; glyph information (font, font size, space width, etc.) is retained.

## 5.26. Environments

It happens frequently that some text should be printed in a certain format regardless of what may be in effect at the time, for example, in a trap invoked macro to print headers and footers. To solve this `gtroff` processes text in *environments*. An environment contains most of the parameters that control text processing. It is possible to switch amongst these environments; by default `gtroff` processes text in environment 0. The following is the information kept in an environment.

- font parameters (size, family, style, glyph height and slant, space and sentence space size)

- page parameters (line length, title length, vertical spacing, line spacing, indentation, line numbering, centering, right-justifying, underlining, hyphenation data)
- fill and adjust mode
- tab stops, tab and leader characters, escape character, no-break and hyphen indicators, margin character data
- partially collected lines
- input traps
- drawing and fill colours

These environments may be given arbitrary names (see [Identifiers](#), for more info). Old versions of `troff` only had environments named '0', '1', and '2'.

`.ev [env]`

`\n[.ev]`

Switch to another environment. The argument *env* is the name of the environment to switch to. With no argument, `gtroff` switches back to the previous environment. There is no limit on the number of named environments; they are created the first time that they are referenced. The `.ev` read-only register contains the name or number of the current environment. This is a string-valued register.

Note that a call to `ev` (with argument) pushes the previously active environment onto a stack. If, say, environments 'foo', 'bar', and 'zap' are called (in that order), the first `ev` request without parameter switches back to environment 'bar' (which is popped off the stack), and a second call switches back to environment 'foo'.

Here is an example:

```
.ev footnote-env
.fam N
.ps 6
.vs 8
.ll -.5i
.ev

...

.ev footnote-env
\dg Note the large, friendly letters.
.ev
```

`.evc env`

Copy the environment *env* into the current environment.

The following environment data is not copied:

- Partially filled lines.
- The status whether the previous line was interrupted.
- The number of lines still to center, or to right-justify, or to underline (with or without underlined spaces); they are set to zero.

- The status whether a temporary indentation is active.
- Input traps and its associated data.
- Line numbering mode is disabled; it can be reactivated with `‘.nm +0’`.
- The number of consecutive hyphenated lines (set to zero).

`\n[.w]`

`\n[.cht]`

`\n[.cdp]`

`\n[.csk]`

The `\n[.w]` register contains the width of the last glyph added to the current environment.

The `\n[.cht]` register contains the height of the last glyph added to the current environment.

The `\n[.cdp]` register contains the depth of the last glyph added to the current environment. It is positive for glyphs extending below the baseline.

The `\n[.csk]` register contains the *skew* (how far to the right of the glyph's center that `gtroff` should place an accent) of the last glyph added to the current environment.

`\n[.n]`

The `\n[.n]` register contains the length of the previous output line in the current environment.

## 5.27. Suppressing output

`\Onum`

Disable or enable output depending on the value of *num*:

`‘\O0’` Disable any glyphs from being emitted to the device driver, provided that the escape occurs at the outer level (see `\O[3]` and `\O[4]`). Motion is not suppressed so effectively `\O[0]` means *pen up*.

`‘\O1’` Enable output of glyphs, provided that the escape occurs at the outer level.

`\O0` and `\O1` also reset the four registers `‘opminx’`, `‘opminy’`, `‘opmaxx’`, and `‘opmaxy’` to `-1`. See [Register Index](#). These four registers mark the top left and bottom right hand corners of a box that encompasses all written glyphs.

For example the input text:

```
Hello \O[0]world \O[1]this is a test.
```

produces the following output:

```
Hello this is a test.
```

`‘\O2’` Provided that the escape occurs at the outer level, enable output of glyphs and also write out to `stderr` the page number and four registers encompassing the glyphs previously written since the last call to `\O`.

- `\O3` Begin a nesting level. At start-up, `gtroff` is at outer level. The current level is contained within the read-only register `.O`. See [Built-in Registers](#).
- `\O4` End a nesting level. The current level is contained within the read-only register `.O`. See [Built-in Registers](#).
- `\O[5Pfilename]`  
This escape is `grohtml` specific. Provided that this escape occurs at the outer nesting level write the `filename` to `stderr`. The position of the image, *P*, must be specified and must be one of `l`, `r`, `c`, or `i` (left, right, centered, inline). *filename* is associated with the production of the next inline image.

## 5.28. Colors

`.color [n]`

`\n[.color]`

If *n* is missing or non-zero, activate colors (this is the default); otherwise, turn it off.

The read-only number register `.color` is 1 if colors are active, 0 otherwise.

Internally, `color` sets a global flag; it does not produce a token. Similar to the `cp` request, you should use it at the beginning of your document to control color output.

Colors can be also turned off with the `-c` command-line option.

`.defcolor ident scheme color_components`

Define color with name *ident*. *scheme* can be one of the following values: `rgb` (three components), `cmY` (three components), `cmYk` (four components), and `gray` or `grey` (one component).

Color components can be given either as a hexadecimal string or as positive decimal integers in the range 0--65535. A hexadecimal string contains all color components concatenated. It must start with either `#` or `##`; the former specifies hex values in the range 0--255 (which are internally multiplied by 257), the latter in the range 0--65535. Examples: `#FFC0CB` (pink), `##ffffff0000ffff` (magenta). The default color name `@c{default}` can't be redefined; its value is device-specific (usually black). It is possible that the default color for `\m` and `\M` is not identical.

A new scaling indicator `f` has been introduced, which multiplies its value by 65536; this makes it convenient to specify color components as fractions in the range 0 to 1 (1f equals 65536u). Example:

```
.defcolor darkgreen rgb 0.1f 0.5f 0.2f
```

Note that `f` is the default scaling indicator for the `defcolor` request, thus the above statement is equivalent to

```
.defcolor darkgreen rgb 0.1 0.5 0.2
```

`.gcolor [color]`

`\mC`

`\m(CO`

`\m[color]`

`\n[.m]`

Set (glyph) drawing color. The following examples show how to turn the next four words red.

```
.gcolor red
these are in red
.gcolor
and these words are in black.
\m[red]these are in red\m[] and these words are in black.
```

The escape `\m[]` returns to the previous color, as does a call to `gcolor` without an argument.

The name of the current drawing color is available in the read-only, string-valued number register `'m'`.

The drawing color is associated with the current environment (see [Environments](#)).

Note that `\m` doesn't produce an input token in `gtroff`. As a consequence, it can be used in requests like `mc` (which expects a single character as an argument) to change the color on the fly:

```
.mc \m[red]x\m[]
.fcolor [color]
\Mc
\M(co
\M[color]
\n[.M]
```

Set fill (background) color for filled objects drawn with the `\D' . . . '` commands.

A red ellipse can be created with the following code:

```
\M[red]\h'0.5i'\D'E 2i 1i'\M[]
```

The escape `\M[]` returns to the previous fill color, as does a call to `fcolor` without an argument.

The name of the current fill (background) color is available in the read-only, string-valued number register `'M'`.

The fill color is associated with the current environment (see [Environments](#)).

Note that `\M` doesn't produce an input token in `gtroff`.

## 5.29. I/O

`gtroff` has several requests for including files:

```
.so file
```

Read in the specified *file* and includes it in place of the `so` request. This is quite useful for large documents, e.g. keeping each chapter in a separate file. See [gsoelim](#), for more information.

Since `gtroff` replaces the `so` request with the contents of *file*, it makes a difference whether the data is terminated with a newline or not: Assuming that file `xxx` contains the word 'foo' without a final newline, this

```
This is
.so xxx
bar
```

yields ‘This is foobar’.

The search path for *file* can be controlled with the `-I` command-line option.

`.pso` *command*

Read the standard output from the specified *command* and includes it in place of the `pso` request.

This request causes an error if used in safer mode (which is the default). Use `gtrouff`’s or `trouff`’s `-U` option to activate unsafe mode.

The comment regarding a final newline for the `so` request is valid for `pso` also.

`.mso` *file*

Identical to the `so` request except that `gtrouff` searches for the specified *file* in the same directories as macro files for the `-m` command-line option. If the file name to be included has the form *name.tmac* and it isn’t found, `mso` tries to include `tmac.name` and vice versa. If the file does not exist, a warning of type ‘file’ is emitted. See [Debugging](#), for information about warnings.

`.trf` *file*

`.cf` *file*

Transparently output the contents of *file*. Each line is output as if it were preceded by `\!`; however, the lines are *not* subject to copy mode interpretation. If the file does not end with a newline, then a newline is added (`trf` only). For example, to define a macro `x` containing the contents of file `f`, use

```
.ev 1
.di x
.trf f
.di
.ev
```

The calls to `ev` prevent that the current partial input line becomes part of the diversion.

Both `trf` and `cf`, when used in a diversion, embeds an object in the diversion which, when reread, causes the contents of *file* to be transparently copied through to the output. In UNIX `trouff`, the contents of *file* is immediately copied through to the output regardless of whether there is a current diversion; this behaviour is so anomalous that it must be considered a bug.

While `cf` copies the contents of *file* completely unprocessed, `trf` disallows characters such as NUL that are not valid `gtrouff` input characters (see [Identifiers](#)).

For `cf`, within a diversion, ‘completely unprocessed’ means that each line of a file to be inserted is handled as if it were preceded by `\!\!`.

Both requests cause a line break.

`.nx` [*file*]

Force `gtrouff` to continue processing of the file specified as an argument. If no argument is given, immediately jump to the end of file.

`.rd` [*prompt* [*arg1 arg2 ...*]]

Read from standard input, and include what is read as though it were part of the input file. Text is read until a blank line is encountered.

If standard input is a TTY input device (keyboard), write *prompt* to standard error,

followed by a colon (or send BEL for a beep if no argument is given).

Arguments after *prompt* are available for the input. For example, the line

```
.rd data foo bar
```

with the input 'This is \\$.2.' prints

```
This is bar.
```

Using the `nx` and `rd` requests, it is easy to set up form letters. The form letter template is constructed like this, putting the following lines into a file called `repeat.let`:

```
.ce
*(td
.sp 2
.nf
.rd
.sp
.rd
.fi
Body of letter.
.bp
.nx repeat.let
```

When this is run, a file containing the following lines should be redirected in. Note that requests included in this file are executed as though they were part of the form letter. The last block of input is the `ex` request, which tells `groff` to stop processing. If this was not there, `groff` would not know when to stop.

```
Trent A. Fisher
708 NW 19th Av., #202
Portland, OR 97209
```

```
Dear Trent,
```

```
Len Adollar
4315 Sierra Vista
San Diego, CA 92103
```

```
Dear Mr. Adollar,
```

```
.ex
```

`.pi` *pipe*

Pipe the output of `gtroff` to the shell command(s) specified by *pipe*. This request must occur before `gtroff` has a chance to print anything.

`pi` causes an error if used in safer mode (which is the default). Use `groff`'s or `troff`'s `-U` option to activate unsafe mode.

Multiple calls to `pi` are allowed, acting as a chain. For example,

```
.pi foo
.pi bar
...
```

is the same as '`.pi foo | bar`'.

Note that the intermediate output format of `gtroff` is piped to the specified commands. Consequently, calling `groff` without the `-z` option normally causes a fatal error.

`.sy cmds`

`\n[systat]`

Execute the shell command(s) specified by *cmds*. The output is not saved anywhere, so it is up to the user to do so.

This request causes an error if used in safer mode (which is the default). Use `groff's` or `troff's` `-U` option to activate unsafe mode.

For example, the following code fragment introduces the current time into a document:

```
.sy perl -e 'printf ".nr H %d\n.nr M %d\n.nr S %d\n", \
 (localtime(time))[2,1,0]' > /tmp/x\n[$$]
.so /tmp/x\n[$$]
.sy rm /tmp/x\n[$$]
\nH:\nM:\nS
```

Note that this works by having the `perl` script (run by `sy`) print out the `nr` requests that set the number registers `H`, `M`, and `S`, and then reads those commands in with the `so` request.

For most practical purposes, the number registers `seconds`, `minutes`, and `hours`, which are initialized at start-up of `gtroff`, should be sufficient. Use the `af` request to get a formatted output:

```
.af hours 00
.af minutes 00
.af seconds 00
\n[hours]:\n[minutes]:\n[seconds]
```

The `systat` read-write number register contains the return value of the `system()` function executed by the last `sy` request.

`.open stream file`

`.opena stream file`

Open the specified *file* for writing and associates the specified *stream* with it.

The `opena` request is like `open`, but if the file exists, append to it instead of truncating it.

Both `open` and `opena` cause an error if used in safer mode (which is the default). Use `groff's` or `troff's` `-U` option to activate unsafe mode.

`.write stream data`

`.writec stream data`

Write to the file associated with the specified *stream*. The stream must previously have been the subject of an `open` request. The remainder of the line is interpreted as the `ds` request reads its second argument: A leading `""` is stripped, and it is read in copy-in mode.

The `writec` request is like `write`, but only `write` appends a newline to the data.

`.writem stream xx`

Write the contents of the macro or string *xx* to the file associated with the specified

*stream*.

*xx* is read in copy mode, i.e., already formatted elements are ignored. Consequently, diversions must be unformatted with the `asciify` request before calling `writem`. Usually, this means a loss of information.

`.close stream`

Close the specified *stream*; the stream is no longer an acceptable argument to the `write` request.

Here a simple macro to write an index entry.

```
.open idx test.idx
.
.de IX
. write idx \n[%] \${*}
..
.
.IX test entry
.
.close idx
```

`\ve`

`\V(ev`

`\V[env]`

Interpolate the contents of the specified environment variable *env* (one-character name *e*, two-character name *ev*) as returned by the function `getenv`. `\V` is interpreted in copy-in mode.

### 5.30. Postprocessor Access

There are two escapes that give information directly to the postprocessor. This is particularly useful for embedding `POSTSCRIPT` into the final document.

`.device xxx`

`\X'xxx'`

Embeds its argument into the `gtroff` output preceded with 'x X'.

The escapes `\&`, `\)`, `\%`, and `\:` are ignored within `\X`, `\'` and `\~` are converted to single space characters. All other escapes (except `\`, which produces a backslash) cause an error.

Contrary to `\X`, the `device` request simply processes its argument in copy mode (see [Copy-in Mode](#)).

If the 'use\_charnames\_in\_special' keyword is set in the `DESC` file, special characters no longer cause an error; they are simply output verbatim. Additionally, the backslash is represented as `\.`

only.

`.devicem xx`

`\Yn`

`\Y(nm`

`\Y[name]`

This is approximately equivalent to `"\X'\*[name]"` (one-character name *n*, two-character name *nm*). However, the contents of the string or macro *name* are not interpreted;

also it is permitted for *name* to have been defined as a macro and thus contain newlines (it is not permitted for the argument to `\X` to contain newlines). The inclusion of newlines requires an extension to the UNIX `tgroff` output format, and confuses drivers that do not know about this extension (see [Device Control Commands](#)).

See [Output Devices](#).

### 5.31. Miscellaneous

This section documents parts of `gtroff` that cannot (yet) be categorized elsewhere in this manual.

```
.nm [start [inc [space [indent]]]]
```

Print line numbers. *start* is the line number of the *next* output line. *inc* indicates which line numbers are printed. For example, the value 5 means to emit only line numbers that are multiples of 5; this defaults to 1. *space* is the space to be left between the number and the text; this defaults to one digit space. The fourth argument is the indentation of the line numbers, defaulting to zero. Both *space* and *indent* are given as multiples of digit spaces; they can be negative also. Without any arguments, line numbers are turned off.

`gtroff` reserves three digit spaces for the line number (which is printed right-justified) plus the amount given by *indent*; the output lines are concatenated to the line numbers, separated by *space*, and *without* reducing the line length. Depending on the value of the horizontal page offset (as set with the `po` request), line numbers that are longer than the reserved space stick out to the left, or the whole line is moved to the right.

Parameters corresponding to missing arguments are not changed; any non-digit argument (to be more precise, any argument starting with a character valid as a delimiter for identifiers) is also treated as missing.

If line numbering has been disabled with a call to `nm` without an argument, it can be reactivated with `nm +0`, using the previously active line numbering parameters.

The parameters of `nm` are associated with the current environment (see [Environments](#)). The current output line number is available in the number register `ln`.

```
.po 1m
.ll 2i
This test shows how line numbering works with groff.
.nm 999
This test shows how line numbering works with groff.
.br
.nm xxx 3 2
.ll -\w'0'u
This test shows how line numbering works with groff.
.nn 2
This test shows how line numbering works with groff.
```

And here the result:

```
This test shows how
line numbering works
999 with groff. This
```

```
1000 test shows how line
1001 numbering works with
1002 groff.
```

```
 This test shows how
 line numbering
works with groff.
```

```
 This test shows how
1005 line numbering
works with groff.
```

**.nn** *[skip]*

Temporarily turn off line numbering. The argument is the number of lines not to be numbered; this defaults to 1.

**.mc** *glyph [dist]*

Print a *margin character* to the right of the text.<sup>24</sup> The first argument is the glyph to be printed. The second argument is the distance away from the right margin. If missing, the previously set value is used; default is 10 pt). For text lines that are too long (that is, longer than the text length plus *dist*), the margin character is directly appended to the lines.

With no arguments the margin character is turned off. If this occurs before a break, no margin character is printed.

For compatibility with AT&T `troff`, a call to `mc` to set the margin character can't be undone immediately; at least one line gets a margin character. Thus

```
.ll 1i
.mc \[br]
.mc
xxx
.br
xxx
```

produces

```
xxx |
xxx
```

For empty lines and lines produced by the `tl` request no margin character is emitted. The margin character is associated with the current environment (see [Environments](#)). This is quite useful for indicating text that has changed, and, in fact, there are programs available for doing this (they are called `nrchbar` and `changebar` and can be found in any 'comp.sources.unix' archive).

```
.ll 3i
.mc |
This paragraph is highlighted with a margin
character.
.sp
Note that vertical space isn't marked.
.br
\&
```

---

<sup>24</sup> *Margin character* is a misnomer since it is an output glyph.

```
.br
But we can fake it with '\&'.
```

**Result:**

```
This paragraph is highlighted |
with a margin character. |
```

```
Note that vertical space isn't |
marked. |
```

```
But we can fake it with '\&' . |
```

```
.psbb filename
```

```
\n[llx]
```

```
\n[lly]
```

```
\n[urx]
```

```
\n[ury]
```

Retrieve the bounding box of the `POSTSCRIPT` image found in *filename*. The file must conform to Adobe's *Document Structuring Conventions* (DSC); the command searches for a `%%BoundingBox` comment and extracts the bounding box values into the number registers `llx`, `lly`, `urx`, and `ury`. If an error occurs (for example, `psbb` cannot find the `%%BoundingBox` comment), it sets the four number registers to zero.

The search path for *filename* can be controlled with the `-I` command-line option.

### 5.32. gtroff Internals

`gtroff` processes input in three steps. One or more input characters are converted to an *input token*.<sup>25</sup> Then, one or more input tokens are converted to an *output node*. Finally, output nodes are converted to the intermediate output language understood by all output devices.

Actually, before step one happens, `gtroff` converts certain escape sequences into reserved input characters (not accessible by the user); such reserved characters are used for other internal processing also -- this is the very reason why not all characters are valid input. See [Identifiers](#), for more on this topic.

For example, the input string `'fi[:u]'` is converted into a character token `'f'`, a character token `'i'`, and a special token `':u'` (representing u umlaut). Later on, the character tokens `'f'` and `'i'` are merged to a single output node representing the ligature glyph `'fi'` (provided the current font has a glyph for this ligature); the same happens with `':u'`. All output glyph nodes are 'processed', which means that they are invariably associated with a given font, font size, advance width, etc. During the formatting process, `gtroff` itself adds various nodes to control the data flow.

Macros, diversions, and strings collect elements in two chained lists: a list of input tokens that have been passed unprocessed, and a list of output nodes. Consider the following the diversion.

```
.di xxx
a
```

<sup>25</sup> Except the escapes `\f`, `\F`, `\H`, `\m`, `\M`, `\R`, `\s`, and `\S`, which are processed immediately if not in copy-in mode.

```

\!b
c
.br
.di

```

It contains these elements.

| node list                 | token list | element number |
|---------------------------|------------|----------------|
| <i>line start node</i>    | ---        | 1              |
| <i>glyph node a</i>       | ---        | 2              |
| <i>word space node</i>    | ---        | 3              |
| ---                       | b          | 4              |
| ---                       | \n         | 5              |
| <i>glyph node c</i>       | ---        | 6              |
| <i>vertical size node</i> | ---        | 7              |
| <i>vertical size node</i> | ---        | 8              |
| ---                       | \n         | 9              |

Elements 1, 7, and 8 are inserted by `gtroff`; the latter two (which are always present) specify the vertical extent of the last line, possibly modified by `\x`. The `br` request finishes the current partial line, inserting a newline input token, which is subsequently converted to a space when the diversion is reread. Note that the word space node has a fixed width that isn't stretchable anymore. To convert horizontal space nodes back to input tokens, use the `unformat` request.

Macros only contain elements in the token list (and the node list is empty); diversions and strings can contain elements in both lists.

Note that the `chop` request simply reduces the number of elements in a macro, string, or diversion by one. Exceptions are *compatibility save* and *compatibility ignore* input tokens, which are ignored. The `substring` request also ignores those input tokens.

Some requests like `tr` or `cflags` work on glyph identifiers only; this means that the associated glyph can be changed without destroying this association. This can be very helpful for substituting glyphs. In the following example, we assume that glyph 'foo' isn't available by default, so we provide a substitution using the `fchar` request and map it to input character 'x'.

```

.fchar \[foo] foo
.tr x \[foo]

```

Now let us assume that we install an additional special font 'bar' that has glyph 'foo'.

```

.special bar
.rchar \[foo]

```

Since glyphs defined with `fchar` are searched before glyphs in special fonts, we must call `rchar` to remove the definition of the fallback glyph. Anyway, the translation is still active; 'x' now maps to the real glyph 'foo'.

Macro and request arguments preserve the compatibility mode:

```

.cp 1 \" switch to compatibility mode
.de xx
\ $1
..

```

```
.cp 0 \" switch compatibility mode off
.xx caf\['e]
=> café
```

Since compatibility mode is on while `de` is called, the macro `xx` activates compatibility mode while executing. Argument `$1` can still be handled properly because it inherits the compatibility mode status which was active at the point where `xx` is called.

After expansion of the parameters, the compatibility save and restore tokens are removed.

### 5.33. Debugging

`gtroff` is not easy to debug, but there are some useful features and strategies for debugging.

```
.lf line [filename]
```

Change the line number and optionally the file name `gtroff` shall use for error and warning messages. *line* is the input line number of the *next* line.

Without argument, the request is ignored.

This is a debugging aid for documents that are split into many files, then put together with `soelim` and other preprocessors. Usually, it isn't invoked manually.

Note that other `troff` implementations (including the original AT&T version) handle `lf` differently. For them, *line* changes the line number of the *current* line.

```
.tm string
```

```
.tm1 string
```

```
.tmc string
```

Send *string* to the standard error output; this is very useful for printing debugging messages among other things.

*string* is read in copy mode.

The `tm` request ignores leading spaces of *string*; `tm1` handles its argument similar to the `ds` request: a leading double quote in *string* is stripped to allow initial blanks.

The `tmc` request is similar to `tm1` but does not append a newline (as is done in `tm` and `tm1`).

```
.ab [string]
```

Similar to the `tm` request, except that it causes `gtroff` to stop processing. With no argument it prints 'User Abort.' to standard error.

```
.ex
```

The `ex` request also causes `gtroff` to stop processing; see also [I/O](#).

When doing something involved it is useful to leave the debugging statements in the code and have them turned on by a command-line flag.

```
.if \n(DB .tm debugging output
```

To activate these statements say

```
groff -rDB=1 file
```

If it is known in advance that there are many errors and no useful output, `gtroff` can be forced to suppress formatted output with the `-z` flag.

`.pev`

Print the contents of the current environment and all the currently defined environments (both named and numbered) on `stderr`.

`.pm`

Print the entire symbol table on `stderr`. Names of all defined macros, strings, and diversions are printed together with their size in bytes. Since `gtroff` sometimes adds nodes by itself, the returned size can be larger than expected.

This request differs from UNIX `troff`: `gtroff` reports the sizes of diversions, ignores an additional argument to print only the total of the sizes, and the size isn't returned in blocks of 128 characters.

`.pnr`

Print the names and contents of all currently defined number registers on `stderr`.

`.ptr`

Print the names and positions of all traps (not including input line traps and diversion traps) on `stderr`. Empty slots in the page trap list are printed as well, because they can affect the priority of subsequently planted traps.

`.fl`

Instruct `gtroff` to flush its output immediately. The intent is for interactive use, but this behaviour is currently not implemented in `gtroff`. Contrary to UNIX `troff`, TTY output is sent to a device driver also (`grotty`), making it non-trivial to communicate interactively.

This request causes a line break.

`.backtrace`

Print a backtrace of the input stack to the standard error stream.

Consider the following in file `test`:

```
.de xxx
. backtrace
..
.de yyy
. xxx
..
.
.YYY
```

On execution, `gtroff` prints the following:

```
test:2: backtrace: macro `xxx'
test:5: backtrace: macro `yyy'
test:8: backtrace: file `test'
```

The option `-b` of `gtroff` internally calls a variant of this request on each error and warning.

`\n[slimit]`

Use the `slimit` number register to set the maximum number of objects on the input stack. If `slimit` is less than or equal to 0, there is no limit set. With no limit, a buggy recursive macro can exhaust virtual memory.

The default value is 1000; this is a compile-time constant.

`.warnscale si`

Set the scaling indicator used in warnings to *si*. Valid values for *si* are ‘u’, ‘i’, ‘c’, ‘p’, and ‘P’. At startup, it is set to ‘i’.

`.spreadwarn [limit]`

Make `gtrouff` emit a warning if the additional space inserted for each space between words in an output line is larger or equal to *limit*. A negative value is changed to zero; no argument toggles the warning on and off without changing *limit*. The default scaling indicator is ‘m’. At startup, `spreadwarn` is deactivated, and *limit* is set to 3 m.

For example,

```
.spreadwarn 0.2m
```

causes a warning if `gtrouff` must add 0.2 m or more for each interword space in a line.

This request is active only if text is justified to both margins (using ‘.ad b’).

`gtrouff` has command-line options for printing out more warnings (`-w`) and for printing backtraces (`-b`) when a warning or an error occurs. The most verbose level of warnings is `-ww`.

`.warn [flags]`

`\n[.warn]`

Control the level of warnings checked for. The *flags* are the sum of the numbers associated with each warning that is to be enabled; all other warnings are disabled. The number associated with each warning is listed below. For example, `.warn 0` disables all warnings, and `.warn 1` disables all warnings except that about missing glyphs. If no argument is given, all warnings are enabled.

The read-only number register `.warn` contains the current warning level.

### 5.33.1. Warnings

The warnings that can be given to `gtrouff` are divided into the following categories. The name associated with each warning is used by the `-w` and `-W` options; the number is used by the `warn` request and by the `.warn` register.

‘char’

‘1’ Non-existent glyphs.<sup>26</sup> This is enabled by default.

‘number’

‘2’ Invalid numeric expressions. This is enabled by default. See [Expressions](#).

‘break’

‘4’ In fill mode, lines that could not be broken so that their length was less than the line length. This is enabled by default.

‘delim’

‘8’ Missing or mismatched closing delimiters.

---

<sup>26</sup> `char` is a misnomer since it reports missing glyphs -- there aren’t missing input characters, only invalid ones.

- 'el'
  - '16' Use of the `el` request with no matching `ie` request. See [if-else](#).
- 'scale'
  - '32' Meaningless scaling indicators.
- 'range'
  - '64' Out of range arguments.
- 'syntax'
  - '128' Dubious syntax in numeric expressions.
- 'di'
  - '256' Use of `di` or `da` without an argument when there is no current diversion.
- 'mac'
  - '512' Use of undefined strings, macros and diversions. When an undefined string, macro, or diversion is used, that string is automatically defined as empty. So, in most cases, at most one warning is given for each name.
- 'reg'
  - '1024' Use of undefined number registers. When an undefined number register is used, that register is automatically defined to have a value of 0. So, in most cases, at most one warning is given for use of a particular name.
- 'tab'
  - '2048' Use of a tab character where a number was expected.
- 'right-brace'
  - '4096' Use of `\}` where a number was expected.
- 'missing'
  - '8192' Requests that are missing non-optional arguments.
- 'input'
  - '16384' Invalid input characters.
- 'escape'
  - '32768' Unrecognized escape sequences. When an unrecognized escape sequence `\X` is encountered, the escape character is ignored, and `X` is printed.
- 'space'
  - '65536' Missing space between a request or macro and its argument. This warning is given when an undefined name longer than two characters is encountered, and the first two characters of the name make a defined name. The request or macro is not invoked. When this warning is given, no macro is automatically defined. This is enabled by default. This warning never occurs in compatibility mode.

‘font’

‘131072’ Non-existent fonts. This is enabled by default.

‘ig’

‘262144’ Invalid escapes in text ignored with the `ig` request. These are conditions that are errors when they do not occur in ignored text.

‘color’

‘524288’ Color related warnings.

‘file’

‘1048576’

Missing files. The `ms0` request gives this warning when the requested macro file does not exist. This is enabled by default.

‘all’

All warnings except ‘di’, ‘mac’ and ‘reg’. It is intended that this covers all warnings that are useful with traditional macro packages.

‘w’

All warnings.

### 5.34. Implementation Differences

GNU `troff` has a number of features that cause incompatibilities with documents written with old versions of `troff`.

Long names cause some incompatibilities. UNIX `troff` interprets

```
.dsabcd
```

as defining a string ‘ab’ with contents ‘cd’. Normally, GNU `troff` interprets this as a call of a macro named `dsabcd`. Also UNIX `troff` interprets `\*[` or `\n[` as references to a string or number register called ‘[’. In GNU `troff`, however, this is normally interpreted as the start of a long name. In compatibility mode GNU `troff` interprets long names in the traditional way (which means that they are not recognized as names).

```
.cp [n]
```

```
.do cmd
```

```
\n[.C]
```

If `n` is missing or non-zero, turn on compatibility mode; otherwise, turn it off.

The read-only number register `.C` is 1 if compatibility mode is on, 0 otherwise.

Compatibility mode can be also turned on with the `-C` command-line option.

The `do` request turns off compatibility mode while executing its arguments as a `gtroff` command. However, it does not turn off compatibility mode while processing the macro itself. To do that, use the `del` request (or manipulate the `.C` register manually). See [Writing Macros](#).

```
.do fam T
```

executes the `fam` request when compatibility mode is enabled.

`gtroff` restores the previous compatibility setting before interpreting any files sourced by the `cmd`.

Two other features are controlled by `-C`. If not in compatibility mode, GNU `troff` preserves the input level in delimited arguments:

```
.ds xx '
\w'abc\'(xxdef'
```

In compatibility mode, the string `'72def'` is returned; without `-C` the resulting string is `'168'` (assuming a TTY output device).

Finally, the escapes `\f`, `\H`, `\m`, `\M`, `\R`, `\s`, and `\S` are transparent for recognizing the beginning of a line only in compatibility mode (this is a rather obscure feature). For example, the code

```
.de xx
Hello!
..
\fB.xx\fP
```

prints `'Hello!'` in bold face if in compatibility mode, and `'xx'` in bold face otherwise.

GNU `troff` does not allow the use of the escape sequences `\|`, `\^`, `\&`, `\{`, `\}`, `\SP`, `\'`, `\``, `\-`, `\_`, `\!`, `\%`, and `\c` in names of strings, macros, diversions, number registers, fonts or environments; UNIX `troff` does. The `\A` escape sequence (see [Identifiers](#)) may be helpful in avoiding use of these escape sequences in names.

Fractional point sizes cause one noteworthy incompatibility. In UNIX `troff` the `ps` request ignores scale indicators and thus

```
.ps 10u
```

sets the point size to 10 points, whereas in GNU `troff` it sets the point size to 10 scaled points. See [Fractional Type Sizes](#), for more information.

In GNU `troff` there is a fundamental difference between (unformatted) input characters and (formatted) output glyphs. Everything that affects how a glyph is output is stored with the glyph node; once a glyph node has been constructed it is unaffected by any subsequent requests that are executed, including `bd`, `cs`, `tkf`, `tr`, or `fp` requests. Normally glyphs are constructed from input characters at the moment immediately before the glyph is added to the current output line. Macros, diversions and strings are all, in fact, the same type of object; they contain lists of input characters and glyph nodes in any combination. A glyph node does not behave like an input character for the purposes of macro processing; it does not inherit any of the special properties that the input character from which it was constructed might have had. For example,

```
.di x
\\
.br
.di
.x
```

prints `'\'` in GNU `troff`; each pair of input backslashes is turned into one output backslash and the resulting output backslashes are not interpreted as escape characters when they are reread. UNIX `troff` would interpret them as escape characters when they were reread and would end up printing one `'\'`. The correct way to obtain a printable backslash is to use the `\e` escape sequence: This always prints a single instance of the current escape character, regardless of whether or not it is used in a diversion; it also works in both

GNU `troff` and UNIX `troff`.<sup>27</sup> To store, for some reason, an escape sequence in a diversion that is interpreted when the diversion is reread, either use the traditional `\!` transparent output facility, or, if this is unsuitable, the new `\?` escape sequence.

See [Diversions](#), and [gtroff Internals](#), for more information.

---

<sup>27</sup> To be completely independent of the current escape character, use `\(rs`, which represents a reverse solidus (backslash) glyph.

## 6. Preprocessors

This chapter describes all preprocessors that come with `groff` or which are freely available.

### 6.1. `eqn`

#### NAME

`eqn` – format equations for troff or MathML

#### SYNOPSIS

`eqn [-rvCNR] [-dxy] [-Tname] [-Mdir] [-fF] [-sn] [-pn] [-mn] [files...]`

#### DESCRIPTION

This manual page describes the GNU version of `eqn`, which is part of the `groff` document formatting system. `eqn` compiles descriptions of equations embedded within `troff` input files into commands that are understood by `troff`. Normally, it should be invoked using the `-e` option of `groff`. The syntax is quite compatible with Unix `eqn`. The output of GNU `eqn` cannot be processed with Unix `troff`; it must be processed with GNU `troff`. If no files are given on the command line, the standard input is read. A filename of `-` causes the standard input to be read.

`eqn` searches for the file `eqnrc` in the directories given with the `-M` option first, then in `/usr/lib/groff/site-tmac`, `/usr/share/groff/site-tmac`, and finally in the standard macro directory `/usr/share/groff/1.22.3.rc1.24-ea225/tmac`. If it exists, `eqn` processes it before the other input files. The `-R` option prevents this.

GNU `eqn` does not provide the functionality of `neqn`: it does not support low-resolution, typewriter-like devices (although it may work adequately for very simple input).

#### OPTIONS

It is possible to have whitespace between a command-line option and its parameter.

- `-dxy` Specify delimiters `x` and `y` for the left and right end, respectively, of in-line equations. Any `delim` statements in the source file overrides this.
- `-C` Recognize `.EQ` and `.EN` even when followed by a character other than space or newline. Also, the statement `'delim on'` is not handled specially.
- `-N` Don't allow newlines within delimiters. This option allows `eqn` to recover better from missing closing delimiters.
- `-v` Print the version number.
- `-r` Only one size reduction.
- `-mn` The minimum point-size is `n`. `eqn` does not reduce the size of subscripts or superscripts to a smaller size than `n`.
- `-Tname`

The output is for device `name`. Normally, the only effect of this is to define a macro `name` with a value of `1`; `eqnrc` uses this to provide definitions appropriate for the output device. However, if the specified device is "MathML", the output is MathML markup rather than troff commands, and `eqnrc` is not

loaded at all. The default output device is **ps**.

- Mdir** Search *dir* for **eqnrc** before the default directories.
- R** Don't load **eqnrc**.
- fF** This is equivalent to a **gfont F** command.
- sn** This is equivalent to a **gsize n** command. This option is deprecated. **eqn** normally sets equations at whatever the current point size is when the equation is encountered.
- pn** This says that subscripts and superscripts should be *n* points smaller than the surrounding text. This option is deprecated. Normally **eqn** sets subscripts and superscripts at 70% of the size of the surrounding text.

## USAGE

Only the differences between GNU **eqn** and Unix **eqn** are described here.

GNU **eqn** emits Presentation MathML output when invoked with the **-T MathML** option.

GNU **eqn** sets the input token "... " as three periods or low dots, rather than the three centered dots of classic **eqn**. To get three centered dots, write **cdots** or **cdot cdot cdot**

Most of the new features of the GNU **eqn** input language are based on T<sub>E</sub>X. There are some references to the differences between T<sub>E</sub>X and GNU **eqn** below; these may safely be ignored if you do not know T<sub>E</sub>X.

### Controlling delimiters

If not in compatibility mode, **eqn** recognizes

**delim on**

to restore the delimiters which have been previously disabled with a call to '**delim off**'. If delimiters haven't been specified, the call has no effect.

### Automatic spacing

**eqn** gives each component of an equation a type, and adjusts the spacing between components using that type. Possible types are:

|                  |                                               |
|------------------|-----------------------------------------------|
| ordinary         | an ordinary character such as '1' or 'x';     |
| operator         | a large operator such as 'Σ';                 |
| binary           | a binary operator such as '+';                |
| relation         | a relation such as '=';                       |
| opening          | a opening bracket such as '(';                |
| closing          | a closing bracket such as ')';                |
| punctuation      | a punctuation character such as ',';          |
| inner            | a subformula contained within brackets;       |
| suppress spacing | that suppresses automatic spacing adjustment. |

Components of an equation get a type in one of two ways.

**type** *t e*

This yields an equation component that contains *e* but that has type *t*, where *t* is one of the types mentioned above. For example, **times** is defined as

```
type "binary" \(\mu
```

The name of the type doesn't have to be quoted, but quoting protects from macro expansion.

**chartype** *t text*

Unquoted groups of characters are split up into individual characters, and the type of each character is looked up; this changes the type that is stored for each character; it says that the characters in *text* from now on have type *t*. For example,

```
chartype "punctuation" .,:;
```

would make the characters ‘.,;’ have type punctuation whenever they subsequently appeared in an equation. The type *t* can also be **letter** or **digit**; in these cases **chartype** changes the font type of the characters. See the **Fonts** subsection.

**New primitives****big** *e*

Enlarges the expression it modifies; intended to have semantics like CSS ‘large’. In troff output, the point size is increased by 5; in MathML output, the expression uses

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

*e1* **smallover** *e2*

This is similar to **over**; **smallover** reduces the size of *e1* and *e2*; it also puts less vertical space between *e1* or *e2* and the fraction bar. The **over** primitive corresponds to the T<sub>E</sub>X **\over** primitive in display styles; **smallover** corresponds to **\over** in non-display styles.

**vcenter** *e*

This vertically centers *e* about the math axis. The math axis is the vertical position about which characters such as ‘+’ and ‘−’ are centered; also it is the vertical position used for the bar of fractions. For example, **sum** is defined as

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

(Note that **vcenter** is silently ignored when generating MathML.)

*e1* **accent** *e2*

This sets *e2* as an accent over *e1*. *e2* is assumed to be at the correct height for a lowercase letter; *e2* is moved down according to whether *e1* is taller or shorter than a lowercase letter. For example, **hat** is defined as

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

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

*e1* **uaccent** *e2*

This sets *e2* as an accent under *e1*. *e2* is assumed to be at the correct height for a character without a descender; *e2* is moved down if *e1* has a descender. **utilde** is pre-defined using **uaccent** as a tilde accent below the baseline.

**split "text"**

This has the same effect as simply

*text*

but *text* is not subject to macro expansion because it is quoted; *text* is split up and the spacing between individual characters is adjusted.

**nosplit text**

This has the same effect as

"text"

but because *text* is not quoted it is subject to macro expansion; *text* is not split up and the spacing between individual characters is not adjusted.

**e opprime**

This is a variant of **prime** that acts as an operator on *e*. It produces a different result from **prime** in a case such as **A opprime sub 1**: with **opprime** the **1** is tucked under the prime as a subscript to the **A** (as is conventional in mathematical typesetting), whereas with **prime** the **1** is a subscript to the prime character. The precedence of **opprime** is the same as that of **bar** and **under**, which is higher than that of everything except **accent** and **uaccent**. In unquoted text a ' that is not the first character is treated like **opprime**.

**special text e**

This constructs a new object from *e* using a **troff(1)** macro named *text*. When the macro is called, the string **Os** contains the output for *e*, and the number registers **Ow**, **Oh**, **Od**, **Oskern**, and **Oskew** contain the width, height, depth, subscript kern, and skew of *e*. (The *subscript kern* of an object says how much a subscript on that object should be tucked in; the *skew* of an object says how far to the right of the center of the object an accent over the object should be placed.) The macro must modify **Os** so that it outputs the desired result with its origin at the current point, and increase the current horizontal position by the width of the object. The number registers must also be modified so that they correspond to the result.

For example, suppose you wanted a construct that 'cancels' an expression by drawing a diagonal line through it.

```
.EQ
define cancel 'special Ca'
.EN
.de Ca
. ds Os \
\Z'*(Os'\
\v'\n(Odu'\
\D'I \n(Owu -\n(Ohu-\n(Odu'\
\v'\n(Ohu'
..
```

Then you could cancel an expression *e* with **cancel { e }**

Here's a more complicated construct that draws a box round an expression:

```

.EQ
define box 'special Bx'
.EN
.de Bx
. ds 0s \
\Z'h'1n'*(0s'
\Z'
\v'\n(0du+1n'
\D'I \n(0wu+2n 0'
\D'I 0 -\n(0hu-\n(0du-2n'
\D'I -\n(0wu-2n 0'
\D'I 0 \n(0hu+\n(0du+2n'
'\
\h'\n(0wu+2n'
. nr 0w +2n
. nr 0d +1n
. nr 0h +1n
..

```

**space  $n$** 

A positive value of the integer  $n$  (in hundredths of an em) sets the vertical spacing before the equation, a negative value sets the spacing after the equation, replacing the default values. This primitive provides an interface to **groff's** `\x` escape (but with opposite sign).

This keyword has no effect if the equation is part of a **pic** picture.

**Extended primitives**

```

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) increases the vertical spacing between rows, using **groff's** `\x` escape (the value has no effect in MathML mode). Negative values are possible but have no effect. If there is more than a single value given in a matrix, the biggest one is used.

**Customization**

When **eqn** is generating troff markup, the appearance of equations is controlled by a large number of parameters. They have no effect when generating MathML mode, which pushes typesetting and fine motions downstream to a MathML rendering engine. These parameters can be set using the **set** command.

**set  $p$   $n$** 

This sets parameter  $p$  to value  $n$ ;  $n$  is an integer. For example,

**set x\_height 45**

says that **eqn** should assume an x height of 0.45 ems.

Possible parameters are as follows. Values are in units of hundredths of an em unless otherwise stated. These descriptions are intended to be expository rather than definitive.

**minimum\_size**

**eqn** doesn't set anything at a smaller point-size than this. The value is in points.

**fat\_offset**

The **fat** primitive emboldens an equation by overprinting two copies of the equation horizontally offset by this amount. This parameter is not used in MathML mode; instead, fat text uses

```
<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.

**accent\_width**

When **bar** or **under** is applied to a single character, the line is this long. 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.

**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.

**null\_delimiter\_space**

This much horizontal space is inserted on each side of a fraction.

**script\_space**

The width of subscripts and superscripts is increased by this amount.

**thin\_space**

This amount of space is automatically inserted after punctuation characters.

**medium\_space**

This amount of space is automatically inserted on either side of binary operators.

**thick\_space**

This amount of space is automatically inserted on either side of relations.

**x\_height**

The height of lowercase letters without ascenders such as 'x'.

**axis\_height**

The height above the baseline of the center of characters such as '+' and '-'. It is important that this value is correct for the font you are using.

**default\_rule\_thickness**

This should set to the thickness of the `\(ru` character, or the thickness of horizontal lines produced with the `\D` escape sequence.

**num1**

The **over** command shifts up the numerator by at least this amount.

**num2**

The **smallover** command shifts up the numerator by at least this amount.

**denom1**

The **over** command shifts down the denominator by at least this amount.

**denom2**

The **smallover** command shifts down the denominator by at least this amount.

**sup1**

Normally superscripts are shifted up by at least this amount.

**sup2**

Superscripts within superscripts or upper limits or numerators of **smallover** fractions are shifted up by at least this amount. This is usually less than sup1.

**sup3**

Superscripts within denominators or square roots or subscripts or lower limits are shifted up by at least this amount. This is usually less than sup2.

**sub1**

Subscripts are normally shifted down by at least this amount.

**sub2**

When there is both a subscript and a superscript, the subscript is shifted down by at least this amount.

**sup\_drop**

The baseline of a superscript is no more than this much amount below the top of the object on which the superscript is set.

**sub\_drop**

The baseline of a subscript is at least this much below the bottom of the object on which the subscript is set.

**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.

**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.

**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.

**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.

**big\_op\_spacing5**

This much vertical space is added above and below limits.

**baseline\_sep**

The baselines of the rows in a pile or matrix are normally this far apart. In most cases this should be 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. In most cases this should be equal to **axis\_height**.

**column\_sep**

This much space is added between columns in a matrix.

**matrix\_side\_sep**

This much space is added at each side of a matrix.

**draw\_lines**

If this is non-zero, lines are drawn using the **\D** escape sequence, rather than with the **\l** escape sequence and the **\(ru** character.

**body\_height**

The amount by which the height of the equation exceeds this is added as extra space before the line containing the equation (using **\x**). The default value is 85.

**body\_depth**

The amount by which the depth of the equation exceeds this is added as extra space after the line containing the equation (using **\x**). The default value is 35.

**nroff**

If this is non-zero, then **ndefine** behaves like **define** and **tdefine** is ignored, otherwise **tdefine** behaves like **define** and **ndefine** is ignored. The default value is 0 (This is typically changed to 1 by the **eqnrc** file for the **ascii**, **latin1**, **utf8**, and **cp1047** devices.)

A more precise description of the role of many of these parameters can be found in Appendix H of *The T<sub>E</sub>Xbook*.

**Macros**

Macros can take arguments. In a macro body, **\$n** where *n* is between 1 and 9, is replaced by the *n*-th argument if the macro is called with arguments; if there are fewer than *n* arguments, it is replaced by nothing. A word containing a left parenthesis where the part of the word before the left parenthesis has been defined using the **define** command is recognized as a macro call with arguments; characters following the left parenthesis up to a matching right parenthesis are treated as comma-

separated arguments; commas inside nested parentheses do not terminate an argument.

**sdefine** *name X anything X*

This is like the **define** command, but *name* is not recognized if called with arguments.

**include** "*file*"

**copy** "*file*"

Include the contents of *file* (**include** and **copy** are synonyms). Lines of *file* beginning with **.EQ** or **.EN** are ignored.

**ifdef** *name X anything X*

If *name* has been defined by **define** (or has been automatically defined because *name* is the output device) process *anything*; otherwise ignore *anything*. *X* can be any character not appearing in *anything*.

**undef** *name*

Remove definition of *name*, making it undefined.

Besides the macros mentioned above, the following definitions are available: **Alpha**, **Beta**, ..., **Omega** (this is the same as **ALPHA**, **BETA**, ..., **OMEGA**), **Idots** (three dots on the base line), and **dollar**.

## Fonts

**eqn** normally uses at least two fonts to set an equation: an italic font for letters, and a roman font for everything else. The existing **gfont** command changes the font that is used as the italic font. By default this is **I**. The font that is used as the roman font can be changed using the new **grfont** command.

**grfont** *f*

Set the roman font to *f*.

The **italic** primitive uses the current italic font set by **gfont**; the **roman** primitive uses the current roman font set by **grfont**. There is also a new **gbfont** command, which changes the font used by the **bold** primitive. If you only use the **roman**, **italic** and **bold** primitives to change fonts within an equation, you can change all the fonts used by your equations just by using **gfont**, **grfont** and **gbfont** commands.

You can control which characters are treated as letters (and therefore set in italics) by using the **chartype** command described above. A type of **letter** causes a character to be set in italic type. A type of **digit** causes a character to be set in roman type.

## FILES

**/usr/share/groff/1.22.3.rc1.24-ea225/tmac/eqnrc** Initialization file.

## MATHML MODE LIMITATIONS

MathML is designed on the assumption that it cannot know the exact physical characteristics of the media and devices on which it will be rendered. It does not support fine control of motions and sizes to the same degree troff does. Thus:

- \* **eqn** parameters have no effect on the generated MathML.
- \* The **special**, **up**, **down**, **fwd**, and **back** operations cannot be implemented, and yield a MathML '<merror>' message instead.

- \* The **vcenter** keyword is silently ignored, as centering on the math axis is the MathML default.
- \* Characters that **eqn** over troff sets extra large – notably the integral sign – may appear too small and need to have their '`<mstyle>`' wrappers adjusted by hand.

As in its troff mode, **eqn** in MathML mode leaves the **.EQ** and **.EN** delimiters in place for displayed equations, but emits no explicit delimiters around inline equations. They can, however, be recognized as strings that begin with '`<math>`' and end with '`</math>`' and do not cross line boundaries.

See the **BUGS** section for translation limits specific to **eqn**.

## BUGS

Inline equations are set at the point size that is current at the beginning of the input line.

In MathML mode, the **mark** and **lineup** features don't work. These could, in theory, be implemented with '`<maligngroup>`' elements.

In MathML mode, 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.

## SEE ALSO

**groff(1)**, **troff(1)**, **pic(1)**, **groff\_font(5)**, *The T<sub>E</sub>Xbook*

## 6.2. gtbl

### NAME

tbl – format tables for troff

### SYNOPSIS

**tbl** [-Cv] [*files* ...]

### DESCRIPTION

This manual page describes the GNU version of **tbl**, which is part of the groff document formatting system. **tbl** compiles descriptions of tables embedded within **troff** input files into commands that are understood by **troff**. Normally, it should be invoked using the **-t** option of **groff**. It is highly compatible with Unix **tbl**. The output generated by GNU **tbl** cannot be processed with Unix **troff**; it must be processed with GNU **troff**. If no files are given on the command line or a filename of **-** is given, the standard input is read.

### OPTIONS

- C** Enable compatibility mode to recognize **.TS** and **.TE** even when followed by a character other than space or newline. Leader characters (**\a**) are handled as interpreted.
- v** Print the version number.

### LANGUAGE OVERVIEW

**tbl** expects to find table descriptions wrapped in the **.TS** (table start) and **.TE** (table end) macros. Within each such table sections, another table can be defined by using the request **.T&** before the final command **.TE**. Each table definition has the following structure:

#### *Global options*

This is optional. This table part can use several of these options distributed in 1 or more lines. The *global option part* must always be finished by a "**semicolon ;**".

#### *Table format specification*

This part must be given, it is not optional. It determines the number of columns (cells) of the table. Moreover each cell is classified by being central, left adjusted, or numerical, etc. This specification can have several lines, but must be finished by a **dot .** at the end of the last line. After each cell definition, *column specifiers* can be appended, but that's optional.

Cells are separated by a tab character by default. That can be changed by the *global option* **tbl(c)**, where *c* is an arbitrary character.

### SIMPLE EXAMPLES

The easiest table definition is.

```
.TS
c c c .
This is centered
Well, this also
```

```
.TE
```

By using **c c c**, each cell in the whole table will be centered. The separating character is here the default *tab*.

The result is

```

This is centered
Well, this also

```

This definition is identical to

```
.TS
tab(@);
ccc.
This@is@centered
Well,@this@also
.TE
```

Here, the separating tab character is changed to the letter **@**.

Moreover a title can be added and the centering directions can be changed to many other formats:

```
.TS
tab(@);
c s s
l c n .
Title
left@centers@123
another@number@75
.TE
```

The result is

```

 Title
left centers 123
another number 75

```

Here **l** means *left-justified*, and **n** means *numerical*, which is here *right-justified*.

## USAGE

### Global options

The line immediately following the **.TS** macro may contain any of the following global options (ignoring the case of characters – Unix `tbl` only accepts options with all characters lowercase or all characters uppercase), separated by spaces, tabs, or commas:

#### **allbox**

Enclose each item of the table in a box.

**box** Enclose the table in a box.

#### **center**

Center the table (default is left-justified). The alternative keyword name **centre** is also recognized (this is a GNU `tbl` extension).

#### **decimalpoint(c)**

Set the character to be recognized as the decimal point in numeric columns

(GNU tbl only).

**delim(xy)**

Use *x* and *y* as start and end delimiters for **eqn(1)**.

**doublebox**

Enclose the table in a double box.

**doubleframe**

Same as doublebox (GNU tbl only).

**expand**

Make the table as wide as the current line length (providing a column separation factor). Ignored if one or more 'x' column specifiers are used (see below).

In case the sum of the column widths is larger than the current line length, the column separation factor is set to zero; such tables extend into the right margin, and there is no column separation at all.

**frame**

Same as box (GNU tbl only).

**linesize(n)**

Set lines or rules (e.g. from **box**) in *n*-point type.

**nokeep**

Don't use diversions to prevent page breaks (GNU tbl only). Normally **tbl** attempts to prevent undesirable breaks in boxed tables by using diversions. This can sometimes interact badly with macro packages own use of diversions, when footnotes, for example, are used.

**nospaces**

Ignore leading and trailing spaces in data items (GNU tbl only).

**nowarn**

Turn off warnings related to tables exceeding the current line width (GNU tbl only).

**tab(x)**

Use the character *x* instead of a tab to separate items in a line of input data.

The global options must end with a semicolon. There might be whitespace between an option and its argument in parentheses.

**Table format specification**

After global options come lines describing the format of each line of the table. Each such format line describes one line of the table itself, except that the last format line (which you must end with a period) describes all remaining lines of the table. A single-key character describes each column of each line of the table. Key characters can be separated by spaces or tabs. You may run format specifications for multiple lines together on the same line by separating them with commas.

You may follow each key character with specifiers that determine the font and point size of the corresponding item, that determine column width, inter-column spacing, etc.

The longest format line defines the number of columns in the table; missing format descriptors at the end of format lines are assumed to be **L**. Extra columns in the

data (which have no corresponding format entry) are ignored.

The available key characters are:

**a,A** Center longest line in this column and then left-justifies all other lines in this column with respect to that centered line. The idea is to use such alphabetic subcolumns (hence the name of the key character) in combination with **L**; they are called subcolumns because **A** items are indented by 1n relative to **L** entries.

Example:

```
.TS
tab(;) ;
ln,an.
item one;1
subitem two;2
subitem three;3
.T&
ln,an.
item eleven;11
subitem twentytwo;22
subitem thirtythree;33
.TE
```

Result:

```
item one 1
subitem two 2
subitem three 3
item eleven 11
subitem twentytwo 22
subitem thirtythree 33
```

**c,C** Center item within the column.

**l,L** Left-justify item within the column.

**n,N** Numerically justify item in the column: Units positions of numbers are aligned vertically. If there is one or more dots adjacent to a digit, use the rightmost one for vertical alignment. If there is no dot, use the rightmost digit for vertical alignment; otherwise, center the item within the column. Alignment can be forced to a certain position using '&'; if there is one or more instances of this special (non-printing) character present within the data, use the leftmost one for alignment. Example:

```
.TS
n.
1
1.5
1.5.3
abcde
a\&bcde
.TE
```

Result:

```

1
1.5
1.5.3
abcde
abcde

```

If numerical entries are combined with **L** or **R** entries – this can happen if the table format is changed with **.T&** – center the widest *number* (of the data entered under the **N** specifier regime) relative to the widest **L** or **R** entry, preserving the alignment of all numerical entries. Contrary to **A** type entries, there is no extra indentation.

Using equations (to be processed with **eqn**) within columns which use the **N** specifier is problematic in most cases due to **tbl**'s algorithm for finding the vertical alignment, as described above. Using the global **delim** option, however, it is possible to make **tbl** ignore the data within **eqn** delimiters for that purpose.

- r,R** Right-justify item within the column.
- s,S** Span previous item on the left into this column. Not allowed for the first column.
- ^** Span down entry from previous row in this column. Not allowed for the first row.
- \_,-** Replace this entry with a horizontal line. Note that ‘\_’ and ‘-’ can be used for table fields only, not for column separator lines.
- =** Replace this entry with a double horizontal line. Note that ‘=’ can be used for table fields only, not for column separator lines.
- |** The corresponding column becomes a vertical rule (if two of these are adjacent, a double vertical rule).

A vertical bar to the left of the first key letter or to the right of the last one produces a line at the edge of the table.

To change the data format within a table, use the **.T&** command (at the start of a line). It is followed by format and data lines (but no global options) similar to the **.TS** request.

### Column specifiers

Here are the specifiers that can appear in suffixes to column key letters (in any order):

- b,B** Short form of **fB** (make affected entries bold).
- d,D** Start an item that vertically spans rows, using the ‘^’ column specifier or ‘\^’ data item, at the bottom of its range rather than vertically centering it (GNU **tbl** only).

Example:

```

.TS
tab(;) allbox;
l l
l ld
r ^

```

```

l rd.
0000;foobar
T{
1111
.br
2222
T};foo
r;
T{
3333
.br
4444
T};bar
\^;\^
.TE

```

Result:

0000	foobar
1111 2222	
r	foo
3333 4444	bar

- e,E** Make equally-spaced columns. All columns marked with this specifier get the same width; this happens after the affected column widths have been computed (this means that the largest width value rules).
- f,F** Either of these specifiers may be followed by a font name (either one or two characters long), font number (a single digit), or long name in parentheses (the last form is a GNU tbl extension). A one-letter font name must be separated by one or more blanks from whatever follows.
- i,I** Short form of **fi** (make affected entries italic).
- m,M** This is a GNU tbl extension. Either of these specifiers may be followed by a macro name (either one or two characters long), or long name in parentheses. A one-letter macro name must be separated by one or more blanks from whatever follows. The macro which name can be specified here must be defined before creating the table. It is called just before the table's cell text is output. As implemented currently, this macro is only called if block input is used, that is, text between 'T{' and 'T}'. The macro should contain only simple **troff** requests to change the text block formatting, like text adjustment, hyphenation, size, or font. The macro is called *after* other cell modifications like **b**, **f** or **v** are output. Thus the macro can overwrite other modification specifiers.
- p,P** Followed by a number, this does a point size change for the affected fields. If signed, the current point size is incremented or decremented (using a signed number instead of a signed digit is a GNU tbl extension). A point size specifier followed by a column separation number must be separated by one or more blanks.

- t,T** Start an item vertically spanning rows at the top of its range rather than vertically centering it.
- u,U** Move the corresponding column up one half-line.
- v,V** Followed by a number, this indicates the vertical line spacing to be used in a multi-line table entry. If signed, the current vertical line spacing is incremented or decremented (using a signed number instead of a signed digit is a GNU `tbl` extension). A vertical line spacing specifier followed by a column separation number must be separated by one or more blanks. No effect if the corresponding table entry isn't a text block.
- w,W** Minimum column width value. Must be followed either by a `troff(1)` width expression in parentheses or a unitless integer. If no unit is given, en units are used. Also used as the default line length for included text blocks. If used multiple times to specify the width for a particular column, the last entry takes effect.
- x,X** An expanded column. After computing all column widths without an `x` specifier, use the remaining line width for this column. If there is more than one expanded column, distribute the remaining horizontal space evenly among the affected columns (this is a GNU extension). This feature has the same effect as specifying a minimum column width.
- z,Z** Ignore the corresponding column for width-calculation purposes, this is, don't use the fields but only the specifiers of this column to compute its width.

A number suffix on a key character is interpreted as a column separation in en units (multiplied in proportion if the `expand` option is on – in case of overfull tables this might be zero). Default separation is 3n.

The column specifier `x` is mutually exclusive with `e` and `w` (but `e` is not mutually exclusive with `w`); if specified multiple times for a particular column, the last entry takes effect: `x` unsets both `e` and `w`, while either `e` or `w` overrides `x`.

### Table data

The format lines are followed by lines containing the actual data for the table, followed finally by `.TE`. Within such data lines, items are normally separated by tab characters (or the character specified with the `tab` option). Long input lines can be broken across multiple lines if the last character on the line is `\` (which vanishes after concatenation).

Note that `tbl` computes the column widths line by line, applying `\w` on each entry which isn't a text block. As a consequence, constructions like

```
.TS
c, l.
\s[20]MM
MMMM
.TE
```

fail; you must either say

```
.TS
cp20, lp20.
MM
```

```

MMMM
. TE

or

. TS
c, l.
\s[20]MM
\s[20]MMMM
. TE

```

A dot starting a line, followed by anything but a digit is handled as a troff command, passed through without changes. The table position is unchanged in this case.

If a data line consists of only ‘\_’ or ‘=’, a single or double line, respectively, is drawn across the table at that point; if a single item in a data line consists of only ‘\_’ or ‘=’, then that item is replaced by a single or double line, joining its neighbours. If a data item consists only of ‘\\_’ or ‘\=’, a single or double line, respectively, is drawn across the field at that point which does not join its neighbours.

A data item consisting only of ‘\Rx’ (‘x’ any character) is replaced by repetitions of character ‘x’ as wide as the column (not joining its neighbours).

A data item consisting only of ‘\^’ indicates that the field immediately above spans downward over this row.

### Text blocks

A text block can be used to enter data as a single entry which would be too long as a simple string between tabs. It is started with ‘T{’ and closed with ‘T}’. The former must end a line, and the latter must start a line, probably followed by other data columns (separated with tabs or the character given with the **tab** global option).

By default, the text block is formatted with the settings which were active before entering the table, possibly overridden by the **m**, **v**, and **w** tbl specifiers. For example, to make all text blocks ragged-right, insert **.na** right before the starting **.TS** (and **.ad** after the table).

If either ‘w’ or ‘x[*cq*]’ specifiers are not given for *all* columns of a text block span, the default length of the text block (to be more precise, the line length used to process the text block 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 total number of columns in the table. Note, however, that the actual diversion width as returned in register **\n[dl]** is used eventually as the text block width. If necessary, you can also control the text block width with a direct insertion of a **.ll** request right after ‘T{’.

### Miscellaneous

The number register **\n[TW]** holds the table width; it can’t be used within the table itself but is defined right before calling **.TE** so that this macro can make use of it.

**tbl** also defines a macro **.T#** which produces the bottom and side lines of a boxed table. While **tbl I** does call this macro itself at the end of the table, it can be used by macro packages to create boxes for multi-page tables by calling it within the page footer. An example of this is shown by the **-ms** macros which provide this functionality if a table starts with **.TS H** instead of the standard call to the **.TS** macro.

## INTERACTION WITH EQN

**tbl**(1) should always be called before **eqn**(1) (**groff**(1) automatically takes care of the correct order of preprocessors).

## GNU TBL ENHANCEMENTS

There is no limit on the number of columns in a table, nor any limit on the number of text blocks. All the lines of a table are considered in deciding column widths, not just the first 200. Table continuation (**.T&**) lines are not restricted to the first 200 lines.

Numeric and alphabetic items may appear in the same column.

Numeric and alphabetic items may span horizontally.

**tbl** uses register, string, macro and diversion names beginning with the digit **3**. When using **tbl** you should avoid using any names beginning with a **3**.

## GNU TBL WITHIN MACROS

Since **tbl** defines its own macros (right before each table) it is necessary to use an 'end-of-macro' macro. Additionally, the escape character has to be switched off. Here an example.

```
.eo
.de ATABLE ..
.TS
allbox tab(;);
cl.
\ $1; \ $2
.TE
...
.ec
.ATABLE A table
.ATABLE Another table
.ATABLE And "another one"
```

Note, however, that not all features of **tbl** can be wrapped into a macro because **tbl** sees the input earlier than **troff**. For example, number formatting with vertically aligned decimal points fails if those numbers are passed on as macro parameters because decimal point alignment is handled by **tbl** itself: It only sees `\$1`, `\$2`, etc., and therefore can't recognize the decimal point.

## BUGS

You should use **.TS H/.TH** in conjunction with a supporting macro package for *all* multi-page boxed tables. If there is no header that you wish to appear at the top of each page of the table, place the **.TH** line immediately after the format section. Do not enclose a multi-page table within keep/release macros, or divert it in any other way.

A text block within a table must be able to fit on one page.

The **bp** request cannot be used to force a page-break in a multi-page table. Instead, define **BP** as follows

```
.de BP
. ie '\n(.z'' .bp \ $1
```

```
. e1 \!.BP \\$1
..
```

and use **BP** instead of **bp**.

Using `\a` directly in a table to get leaders does not work (except in compatibility mode). This is correct behaviour: `\a` is an **uninterpreted** leader. To get leaders use a real leader, either by using a control A or like this:

```
.ds a \a
.TS
tab(;);
lw(1i) l.
A*a;B
.TE
```

A leading and/or trailing `|` in a format line, such as

```
|l r|.
```

gives output which has a 1n space between the resulting bordering vertical rule and the content of the adjacent column, as in

```
.TS
tab(#);
|l r|.
left column#right column
.TE
```

If it is desired to have zero space (so that the rule touches the content), this can be achieved by introducing extra “dummy” columns, with no content and zero separation, before and/or after, as in

```
.TS
tab(#);
r0|l r0|l.
#left column#right column#
.TE
```

The resulting “dummy” columns are invisible and have zero width; note that such columns usually don’t work with TTY devices.

## REFERENCE

Lesk, M.E.: "TBL – A Program to Format Tables". For copyright reasons it cannot be included in the groff distribution, but copies can be found with a title search on the World Wide Web.

## SEE ALSO

**groff(1)**, **troff(1)**

### 6.3. `pic`

#### NAME

`pic` – compile pictures for troff or TeX

#### SYNOPSIS

```
pic [-nvCSU] [filename ...]
pic -t [-cvzCSU] [filename ...]
```

#### DESCRIPTION

This manual page describes the GNU version of **pic**, which is part of the groff document formatting system. **pic** compiles descriptions of pictures embedded within **troff** or TeX input files into commands that are understood by TeX or **troff**. Each picture starts with a line beginning with **.PS** and ends with a line beginning with **.PE**. Anything outside of **.PS** and **.PE** is passed through without change.

It is the user's responsibility to provide appropriate definitions of the **PS** and **PE** macros. When the macro package being used does not supply such definitions (for example, old versions of `-ms`), appropriate definitions can be obtained with **-mpic**: These will center each picture.

#### OPTIONS

Options that do not take arguments may be grouped behind a single `-`. The special option `--` can be used to mark the end of the options. A filename of `-` refers to the standard input.

- C** Recognize **.PS** and **.PE** even when followed by a character other than space or newline.
- S** Safer mode; do not execute **sh** commands. This can be useful when operating on untrustworthy input (enabled by default).
- U** Unsafe mode; revert the default option **-S**.
- n** Don't use the groff extensions to the troff drawing commands. You should use this if you are using a postprocessor that doesn't support these extensions. The extensions are described in **groff\_out(5)**. The **-n** option also causes **pic** not to use zero-length lines to draw dots in troff mode.
- t** TeX mode.
- 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 `\`. A line beginning with **.ps** is given special treatment: it takes an optional integer argument specifying the line thickness (pen size) in millinches; a missing argument restores the previous line thickness; the default line thickness is 8 millinches. 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.
- v** Print the version number.
- 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. **pic** always does this.

**-T dev**

Generate output for the **troff** device *dev*. This is unnecessary because the **troff** output generated by **pic** is device-independent.

## USAGE

This section describes only the differences between GNU **pic** and the original version of **pic**. Many of these differences also apply to newer versions of Unix **pic**. A complete documentation is available in the file

`/usr/share/doc/groff-1.22.3.rc1.24-ea225/pic.ms`

## T<sub>E</sub>X mode

T<sub>E</sub>X mode is enabled by the `-t` option. In T<sub>E</sub>X mode, **pic** will define a vbox called `\graph` for each picture. Use the `\figname` command to change the name of the vbox. You must yourself print that vbox using, for example, the command

```
\centerline{\box\graph}
```

Actually, since the vbox has a height of zero (it is defined with `\vtop`) this will produce slightly more vertical space above the picture than below it;

```
\centerline{\raise 1em\box\graph}
```

would avoid this.

To make the vbox having a positive height and a depth of zero (as used e.g. by L<sup>A</sup>T<sub>E</sub>X's `graphics.sty`), define the following macro in your document:

```
\def\gpicbox#1{%
 \vbox{\unvbox\csname #1\endcsname\kern 0pt}}
```

Now you can simply say `\gpicbox{graph}` instead of `\box\graph`.

You must use a T<sub>E</sub>X driver that supports the **tpic** specials, version 2.

Lines beginning with `\` are passed through transparently; a `%` is added to the end of the line to avoid unwanted spaces. You can safely use this feature to change fonts or to change the value of `\baselineskip`. Anything else may well produce undesirable results; use at your own risk. Lines beginning with a period are not given any special treatment.

## Commands

**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. If *expr3* is prefixed by `*` then *variable* will instead be multiplied by *expr3*. The value of *expr3* can be negative for the additive case; *variable* is then tested whether it is greater than or equal to *expr2*. For the multiplicative case, *expr3* must be greater than zero. If the constraints aren't met, the loop isn't executed. *X* can be any character not occurring 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 occurring in *if-true*. *Y* can be any character not occurring in *if-false*.

**print** *arg* . . .

Concatenate the arguments and print as a line on stderr. Each *arg* must be an expression, a position, or text. This is useful for debugging.

**command** *arg* . . .

Concatenate the arguments and pass them through as a line to troff or T<sub>E</sub>X. Each *arg* must be an expression, a position, or text. This has a similar effect to a line beginning with . or \, but allows the values of variables to be passed through. For example,

```
.PS
x = 14
command ".ds string x is " x "."
.PE
\[string]
```

prints

```
x is 14.
```

**sh** *X command X*

Pass *command* to a shell. *X* can be any character not occurring in *command*.

**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, lines will be read only until a line the first word of which is *word*; that line will then be discarded. *X* can be any character not occurring in *body*. For example,

```
.PS
copy thru % circle at ($1,$2) % until "END"
1 2
3 4
5 6
END
box
.PE
```

is equivalent to

```
.PS
circle at (1,2)
circle at (3,4)
circle at (5,6)
box
.PE
```

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**.

### reset

**reset** *variable1*[,*variable2* ...

Reset pre-defined variables *variable1*, *variable2* ... to their default values. If no arguments are given, reset all pre-defined variables to their default values. Note that assigning a value to **scale** also causes all pre-defined variables that control dimensions to be reset to their default values times the new value of scale.

**plot** *expr* ["*text*"]

This is a text object which is constructed by using *text* as a format string for `sprintf` with an argument of *expr*. If *text* is omitted a format string of "%g" is used. Attributes can be specified in the same way as for a normal text object. Be very careful that you specify an appropriate format string; **pic** does only very limited checking of the string. This is deprecated in favour of **sprintf**.

*variable* := *expr*

This is similar to = except *variable* must already be defined, and *expr* will be assigned to *variable* without creating a variable local to the current block. (By contrast, = defines the variable in the current block if it is not already defined there, and then changes the value in the current block only.) For example, the following:

```
.PS
x = 3
y = 3
[
 x := 5
 y = 5
]
print x " " y
.PE
```

prints

```
5 3
```

Arguments of the form

*X anything X*

are also allowed to be of the form

{ *anything* }

In this case *anything* can contain balanced occurrences of { and }. Strings may contain *X* or imbalanced occurrences of { and }.

### 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**

means draw a line 2 inches long in the current direction. The 'i' (or 'l') 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.

Scientific notation is allowed for numbers. For example

**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 examined. For example,

**[A: [B: [C: box ]]] with .A.B.C.sw at 1,2  
circle at last [].A.B.C**

is acceptable.

Arcs now have compass points determined by the circle of which the arc is a part.

Circles, ellipses, and arcs can be dotted or dashed. In T<sub>E</sub>X mode splines can be dotted or dashed also.

Boxes can have rounded corners. The **rad** attribute specifies the radius of the quarter-circles at each corner. If **norad** 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. This effectively changes the shape of a box from a rectangle to an arbitrary parallelogram. The **xslanted** and **yslanted** attributes specify the x and y offset of the box's upper right corner from its default position.

The **.PS** line can have a second argument specifying a maximum height for the picture. If the width of zero is specified the width will be ignored in computing the scaling factor for the picture. Note that GNU**pic** will always scale a picture by the same amount vertically as well as horizontally. This is different from the 2.0 **pic** which may scale a picture by a different amount vertically than horizontally if a height is specified.

Each text object has an invisible box associated with it. The compass points of a text object are determined by this box. The implicit motion associated with the object is also determined by this box. The dimensions of this box are taken from the width and height attributes; if the width attribute is not supplied then the width will be taken to be **textwid**; if the height attribute is not supplied then the height will be taken to be the number of text strings associated with the object times **textht**. Initially **textwid** and **textht** have a value of 0.

In (almost all) places where a quoted text string can be used, an expression of the form

```
sprintf("format", arg,...)
```

can also be used; this will produce the arguments formatted according to *format*, which should be a string as described in **printf(3)** appropriate for the number of arguments supplied.

The thickness of the lines used to draw objects is controlled by the **linethick** variable. This gives the thickness of lines in points. A negative value means use the default thickness: in T<sub>E</sub>X output mode, this means use a thickness of 8 millinches; in T<sub>E</sub>X output mode with the **-c** option, this means use the line thickness specified by **.ps** lines; in troff output mode, this means use a thickness proportional to the pointsize. A zero value means draw the thinnest possible line supported by the output device. Initially it has a value of -1. There is also a **thick[ness]** attribute. For example,

```
circle thickness 1.5
```

would draw a circle using a line with a 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]**. This takes an optional argument of an expression with a value between 0 and 1; 0 will fill it with white, 1 with black, values in between with a proportionally gray shade. A value greater than 1 can also be used: this means fill with the shade of gray that is currently being used for text and lines. Normally this will be black, but output devices may provide a mechanism for changing this. Without an argument, then the value of the variable **fillval** will be used. Initially this has a value of 0.5. The invisible attribute does not affect the filling of objects. Any text associated with a filled object will be added after the object has

been filled, so that the text will not be obscured by the filling.

Three additional modifiers are available to specify colored objects: **outline[d]** sets the color of the outline, **shaded** the fill color, and **colo[u]r[ed]** sets both. All three keywords expect a suffix specifying the color, for example

**circle shaded "green" outline "black"**

Currently, color support isn't available in T<sub>E</sub>X mode. Predefined color names for **groff** are in the device macro files, for example **ps.tmac**; additional colors can be defined with the **.defcolor** request (see the manual page of **troff(1)** for more details).

To change the name of the vbox in T<sub>E</sub>X mode, set the pseudo-variable **figname** (which is actually a specially parsed command) within a picture. Example:

```
.PS
figname = foobar;
...
.PE
```

The picture is then available in the box **\foobar**.

**pic** assumes that at the beginning of a picture both glyph and fill color are set to the default value.

Arrow heads will be drawn as solid triangles if the variable **arrowhead** is non-zero and either T<sub>E</sub>X mode is enabled or the **-n** option has not been given. Initially **arrowhead** has a value of 1. Note that solid arrow heads are always filled with the current outline color.

The troff output of **pic** is device-independent. The **-T** option is therefore redundant. All numbers are taken to be in inches; numbers are never interpreted to be in troff machine units.

Objects can have an **aligned** attribute. This will only work if the postprocessor is **grops**, or **gropdf**. Any text associated with an object having the **aligned** attribute will be rotated about the center of the object so that it is aligned in the direction from the start point to the end point of the object. Note that this attribute will have no effect for objects whose start and end points are coincident.

In places where *n***th** is allowed '*expr*'**th** is also allowed. Note that '**th**' is a single token: no space is allowed between the ' and the **th**. For example,

```
for i = 1 to 4 do {
 line from 'i'th box.nw to 'i+1'th box.se
}
```

## CONVERSION

To obtain a stand-alone picture from a **pic** file, enclose your **pic** code with **.PS** and **.PE** requests; **roff** configuration commands may be added at the beginning of the file, but no **roff** text.

It is necessary to feed this file into **groff** without adding any page information, so you must check which **.PS** and **.PE** requests are actually called. For example, the mm macro package adds a page number, which is very annoying. At the moment, calling standard **groff** without any macro package works. Alternatively, you can define your own requests, e.g. to do nothing:

```
.de PS
```

```
..
```

```
.de PE
```

```
..
```

**groff** itself does not provide direct conversion into other graphics file formats. But there are lots of possibilities if you first transform your picture into PostScript® format using the **groff** option **-Tps**. Since this *ps*-file lacks BoundingBox information it is not very useful by itself, but it may be fed into other conversion programs, usually named **ps2other** or **pstother** or the like. Moreover, the PostScript interpreter **ghostscript** (**gs**) has built-in graphics conversion devices that are called with the option

```
gs -sDEVICE=<devname>
```

Call

```
gs --help
```

for a list of the available devices.

An alternative may be to use the **-Tpdf** option to convert your picture directly into **PDF** format. The MediaBox of the file produced can be controlled by passing a **-P-p** papersize to **groff**.

As the Encapsulated PostScript File Format **EPS** is getting more and more important, and the conversion wasn't regarded trivial in the past you might be interested to know that there is a conversion tool named **ps2eps** which does the right job. It is much better than the tool **ps2epsi** packaged with **gs**.

For bitmapped graphic formats, you should use **pstopnm**; the resulting (intermediate) **PNM** file can be then converted to virtually any graphics format using the tools of the **netpbm** package.

## FILES

**/usr/share/groff/1.22.3.rc1.24-ea225/tmac/pic.tmac** Example definitions of the **PS** and **PE** macros.

## SEE ALSO

**troff**(1), **groff\_out**(5), **tex**(1), **gs**(1), **ps2eps**(1), **pstopnm**(1), **ps2epsi**(1), **pnm**(5)

Eric S. Raymond, *Making Pictures With GNU PIC*.

**/usr/share/doc/groff-1.22.3.rc1.24-ea225/pic.ps** (this file, together with its source file, **pic.ms**, is part of the **groff** documentation)

Tpic: Pic for T<sub>E</sub>X

Brian W. Kernighan, *PIC — A Graphics Language for Typesetting (User Manual)* AT&T Bell Laboratories, Computing Science Technical Report No. 116 (revised May, 1991).

**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"*

**BUGS**

Input characters that are invalid for **groff** (i.e., those with code 0, or 013 octal, or between 015 and 037 octal, or between 0200 and 0237 octal) are rejected even in T<sub>E</sub>X mode.

The interpretation of **fillval** is incompatible with the pic in 10th edition Unix, which interprets 0 as black and 1 as white.

PostScript® is a registered trademark of Adobe Systems Incorporation.

## 6.4. `grn`

### NAME

`grn` – groff preprocessor for gremlin files

### SYNOPSIS

`grn` [ **-Cv** ] [ **-Tdev** ] [ **-Mdir** ] [ **-Fdir** ] [ *file...* ]

### DESCRIPTION

`grn` is a preprocessor for including *gremlin* pictures in *groff* 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 commands request a *gremlin* file, and the picture in that file is converted and placed in the *troff* input stream. The **.GS** request may be followed by a C, L, or R to center, left, or right justify the whole *gremlin* picture (default justification is 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.

Please note that currently only the `-me` macro package has support for **.GS**, **.GE**, and **.GF**.

The following command-line options are understood:

- Tdev** Prepare output for printer *dev*. The default device is **ps**. See `groff(1)` for acceptable devices.
- Mdir** Prepend *dir* to the default search path for *gremlin* files. The default path is (in that order) the current directory, the home directory, `/usr/lib/groff/site-tmac`, `/usr/share/groff/site-tmac`, and `/usr/share/groff/1.22.3.rc1.24-ea225/tmac`.
- Fdir** Search *dir* for subdirectories *devname* (*name* is the name of the device) for the **DESC** file before the default font directories `/usr/share/groff/site-font`, `/usr/share/groff/1.22.3.rc1.24-ea225/font`, and `/usr/lib/font`.
- C** Recognize **.GS** and **.GE** (and **.GF**) even when followed by a character other than space or newline.
- v** Print the version number.

It is possible to have whitespace between a command-line option and its parameter.

### GRN COMMANDS

Each input line between **.GS** and **.GE** may have one `grn` command. Commands consist of one or two strings separated by white space, the first string being the command and the second its operand. Commands may be upper or lower case 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 file, but can be used at any time to reset the default settings.

**width** *N*

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

**height** *N*

Forces 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 two **file** commands are given, the second one overrides the first. If *name* doesn't exist, an error message is reported and processing continues from the **.GE** line.

## NOTES ABOUT GROFF

Since *grn* is a preprocessor, it doesn't know about current indents, point sizes, margins, number registers, etc. Consequently, no *troff* input can be placed between the **.GS** and **.GE** requests. However, *gremlin* text is now processed by *troff*, so anything valid in a single line of *troff* input is valid in a line of *gremlin* text (barring '.' directives 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, it is best to run *tbl* before *grn*, *pic*, and/or *ideal* to avoid overworking *tbl*. *Eqn* should always be run last.

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 **-me** macros will ensure proper placement.

*grn* uses *troff's* number 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** request (this is for those who want to rewrite these macros).

## GREMLIN FILE FORMAT

There exist two distinct *gremlin* file formats, the original format from the *AED* graphic terminal version, and the *SUN* or *X11* version. An extension to the *SUN/X11* version allowing reference points with negative coordinates is **not** compatible with the *AED* version. As long as *ag remlin* file does not contain negative coordinates, either format will be read correctly by either version of *gremlin* or *grn*. The other difference to the *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 in Table 1 in each format.

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

Table 1. File examples

- The first line of each *gremlin* file contains either the string **gremlinfile** (*AED* version) 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 (drawing area wider than it is tall, with menu across top). **1** means that *gremlin* will display things in vertical format (drawing area taller than it is wide, with menu on 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 chopped to this limit.

## ELEMENT SPECIFICATIONS

- The first line of each element contains a single decimal number giving the type of the element (*AED* version) or its ASCII name (*SUN/X11* version). See Table 2.

### *gremlin* File Format – Object Type Specification

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

Table 2.  
Type Specifications in *gremlin* Files

- After the object type comes a variable number of lines, each specifying a point used to display the element. Each line contains an x-coordinate 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* version) or a single asterisk, “\*” (*SUN/X11* version).
- 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, one more value, 0, is valid. It specifies a polygon with an invisible border. For text, the brush selects a font as follows:

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

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 “\f1” 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 only used for text elements, and contains the text string. There can be spaces inside the text. For arcs, curves, and vectors, this line of the element contains the string “0”.

## NOTES ON COORDINATES

*gremlin* was designed for *AEDs*, 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 range from 0 to 367. Although you needn’t absolutely stick to this range, you’ll get best 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 format “%f1.2”; it’s probably a good idea to use the same format if you want to modify the *grn* code.

**NOTES ON SUN/X11 COORDINATES**

There is no longer a 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*.

**FILES**

**/usr/share/groff/1.22.3.rc1.24-ea225/font/devname/DESC**

Device description file  
for 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)**

## 6.5. `grap`

A free implementation of `grap`, written by Ted Faber, is available as an extra package from the following address:

<http://www.lunabase.org/~faber/Vault/software/grap/>

## 6.6. `gchem`

### NAME

`chem` – groff preprocessor for producing chemical structure diagrams

### SYNOPSIS

```
chem [option...] [--] [filespec...]
chem -h | --help
chem -v | --version
```

### OPTION USAGE

There are no other options than **-h**, **--help**, **-v**, and **--version**; these options provoke the printing of a version or usage information, respectively, and all *filespec* arguments are ignored. A *filespec* argument is either a file name of an existing file or a minus character **-**, meaning standard input. If no argument is specified then standard input is taken automatically.

### 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.

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.

To actually view this, you must run **chem** followed by **groffer**:

```
chem [file ...] | groffer
```

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 ...
```

## THE 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

This

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

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 create dotted or dashed bonds or to specify a **to** place.

## Rings

There are lots of rings, but only 5 and 6-sided rings get much support. **ring** by itself is a 6-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**, ..., **.Vn**, 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 5-sided rings. **ring5** is pentagonal with a side that matches the 6-sided ring; it has four natural directions. **Aflatring** is a 5-sided ring created by chopping one corner of a 6-sided ring so that it exactly matches the 6-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/1.22.3.rc1.24-ea225/pic/chem.pic**

A collection of *pic* macros needed by **chem**.

**/usr/share/groff/1.22.3.rc1.24-ea225/tmac/pic.tmac**

A macro file which redefines **.PS** and **.PE** to center *pic* diagrams.

**/usr/share/doc/groff-1.22.3.rc1.24-ea225/examples/chem/\*.chem**

Example files for *chem*.

**/usr/share/doc/groff-1.22.3.rc1.24-ea225/examples/chem/122/\*.chem**

Example files from the classical *chem* article *CHEM – A Program for Typesetting Chemical Structure Diagrams* [CSTR #122].

## BUGS

Report bugs to the [bug-groff mailing list](#) Include a complete, self-contained example that will allow the bug to be reproduced, and say which version of *groff* and *chem* you are using. You can get both version numbers by calling **chem --version**.

You can also use the [groff mailing list](#) but you must first subscribe to this list. You can do that by visiting the [groff mailing list web page](#)

See **groff(1)** for information on availability.

## 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* at <http://cm.bell-labs.com/cm/cs/who/bwk/index.html>

## SEE ALSO

**groff(1)**, **pic(1)**, **groffer(1)**.

You can still get the original [chem awk source](#) Its **README** file was used for this manual page.

The other classical document on *chem* is "[CHEM – A Program for Typesetting Chemical Structure Diagrams](#)" [CSTR #122]

## 6.7. `refer`

### NAME

`refer` – preprocess bibliographic references for `groff`

### SYNOPSIS

```
refer [-benvCPRS] [-an] [-cfields] [-fn] [-ifields] [-kfield] [-lm,n] [-pfilename]
 [-sfields] [-tn] [-Bfield.macro] [filename...]
```

### DESCRIPTION

This file documents the GNU version of **refer**, which is part of the `groff` document formatting system. **refer** copies the contents of `filename...` to the standard output, except that lines between `.[` and `.]` are interpreted as citations, and lines between `.R1` and `.R2` are interpreted as commands about how citations are to be processed.

Each citation specifies a reference. The citation can specify a reference that is contained in a bibliographic database by giving a set of keywords that only that reference contains. Alternatively it can specify a reference by supplying a database record in the citation. A combination of these alternatives is also possible.

For each citation, **refer** can produce a mark in the text. This mark consists of some label which can be separated from the text and from other labels in various ways. For each reference it also outputs **groff** commands that can be used by a macro package to produce a formatted reference for each citation. The output of **refer** must therefore be processed using a suitable macro package. The `-ms` and `-me` macros are both suitable. The commands to format a citation's reference can be output immediately after the citation, or the references may be accumulated, and the commands output at some later point. If the references are accumulated, then multiple citations of the same reference will produce a single formatted reference.

The interpretation of lines between `.R1` and `.R2` as commands is a new feature of GNU **refer**. Documents making use of this feature can still be processed by Unix `refer` just by adding the lines

```
.de R1
.ig R2
```

```
..
```

to the beginning of the document. This will cause **troff** to ignore everything between `.R1` and `.R2`. The effect of some commands can also be achieved by options. These options are supported mainly for compatibility with Unix `refer`. It is usually more convenient to use commands.

**refer** generates `.If` lines so that filenames and line numbers in messages produced by commands that read **refer** output will be correct; it also interprets lines beginning with `.If` so that filenames and line numbers in the messages and `.If` lines that it produces will be accurate even if the input has been preprocessed by a command such as `soelim(1)`.

**OPTIONS**

It is possible to have whitespace between a command-line option and its parameter.

Most options are equivalent to commands (for a description of these commands see the **Commands** subsection):

- b**     **no-label-in-text; no-label-in-reference**
- e**     **accumulate**
- n**     **no-default-database**
- C**     **compatible**
- P**     **move-punctuation**
- S**     **label "(A.n|Q) ', ' (D.y|D)"; bracket-label " ( " ) "; "**
- an**    **reverse An**
- cfields**  
          **capitalize *fields***
- fn**    **label %n**
- ifields**  
          **search-ignore *fields***
- k**     **label L~%a**
- kfield** **label *field*~%a**
- l**     **label A.nD.y%a**
- lm**    **label A.nmD.y%a**
- l,n**   **label A.nD.y~n%a**
- lm,n**  **label A.nmD.y~n%a**
- pfilename**  
          **database *filename***
- sspec**  
          **sort *spec***
- tn**    **search-truncate *n***

These options are equivalent to the following commands with the addition that the filenames 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**
- Bfield.macro**  
          **annotate *field macro*; no-label-in-reference**

The following options have no equivalent commands:

- v**     Print the version number.
- R**     Don't recognize lines beginning with **.R1/.R2**.

**USAGE**

### Bibliographic databases

The bibliographic database is a text file consisting of records separated by one or more blank lines. Within each record fields start with a% at the beginning of a line. Each field has a one character name that immediately follows the %. It is best to use only upper and lower case letters for the names of fields. The name of the field should be followed by exactly one space, and then by the contents of the field. Empty fields are ignored. The conventional meaning of each field is as follows:

- %A** The name of an author. If the name contains a title such as **Jr** . at the end, it should be separated from the last name by a comma. There can be multiple occurrences of the **%A** field. The order is significant. It is a good idea always to supply an **%A** field or a **%Q** field.
- %B** For an article that is part of a book, the title of the book.
- %C** The place (city) of publication.
- %D** The date of publication. The year should be specified in full. If the month is specified, the name rather than the number of the month should be used, but only the first three letters are required. It is a good idea always to supply a **%D** field; if the date is unknown, a value such as **in press** or **unknown** can be used.
- %E** For an article that is part of a book, the name of an editor of the book. Where the work has editors and no authors, the names of the editors should be given as **%A** fields and , **(ed)** or , **(eds)** should be appended to the last author.
- %G** US Government ordering number.
- %I** The publisher (issuer).
- %J** For an article in a journal, the name of the journal.
- %K** Keywords to be used for searching.
- %L** Label.
- %N** Journal issue number.
- %O** Other information. This is usually printed at the end of the reference.
- %P** Page number. A range of pages can be specified as *m-n*.
- %Q** The name of the author, if the author is not a person. This will only be used if there are no **%A** fields. There can only be one **%Q** field.
- %R** Technical report number.
- %S** Series name.
- %T** Title. For an article in a book or journal, this should be the title of the article.
- %V** Volume number of the journal or book.
- %X** Annotation.

For all fields except **%A** and **%E**, if there is more than one occurrence of a particular field in a record, only the last such field will be used.

If accent strings are used, they should follow the character to be accented. This means that the **AM** macro must be used with the **-ms** macros. Accent strings should not be quoted: use one \ rather than two.

## Citations

The format of a citation is

```
.[opening-text
 flags keywords
 fields
 .]closing-text
```

The *opening-text*, *closing-text* and *flags* components are optional. Only one of the *keywords* and *fields* components need be specified.

The *keywords* component says to search the bibliographic databases for a reference that contains all the words in *keywords*. It is an error if more than one reference is found.

The *fields* component specifies additional fields to replace or supplement those specified in the reference. When references are being accumulated and the *keywords* component is non-empty, then additional fields should be specified only on the first occasion that a particular reference is cited, and will apply to all citations of that reference.

The *opening-text* and *closing-text* component specifies strings to be used to bracket the label instead of the strings specified in the **bracket-label** command. If either of these components is non-empty, the strings specified in the **bracket-label** command will not be used; this behaviour can be altered using the **[** and **]** flags. Note that leading and trailing spaces are significant for these components.

The *flags* component is a list of non-alphanumeric characters each of which modifies the treatment of this particular citation. Unix refer will treat these flags as part of the *keywords* and so will ignore them since they are non-alphanumeric. The following flags are currently recognized:

- #** This says to use the label specified by the **short-label** command, instead of that specified by the **label** command. If no short label has been specified, the normal label will be used. Typically the short label is used with author-date labels and consists of only the date and possibly a disambiguating letter; the **#** is supposed to be suggestive of a numeric type of label.
- [** Precede *opening-text* with the first string specified in the **bracket-label** command.
- ]** Follow *closing-text* with the second string specified in the **bracket-label** command.

One advantage of using the **[** and **]** flags rather than including the brackets in *opening-text* and *closing-text* is that you can change the style of bracket used in the document just by changing the **bracket-label** command. Another advantage is that sorting and merging of citations will not necessarily be inhibited if the flags are used.

If a label is to be inserted into the text, it will be attached to the line preceding the **.[** line. If there is no such line, then an extra line will be inserted before the **.[** line and a warning will be given.

There is no special notation for making a citation to multiple references. Just use a sequence of citations, one for each reference. Don't put anything between the citations. The labels for all the citations will be attached to the line preceding the first citation. The labels may also be sorted or merged. See the description of the **<>** label

expression, and of the **sort-adjacent-labels** and **abbreviate-label-ranges** command. A label will not be merged if its citation has a non-empty *opening-text* or *closing-text*. However, the labels for a citation using the ] flag and without any *closing-text* immediately followed by a citation using the [ flag and without any *opening-text* may be sorted and merged even though the first citation's *opening-text* or the second citation's *closing-text* is non-empty. (If you wish to prevent this just make the first citation's *closing-text* \&.)

## Commands

Commands are contained between lines starting with **.R1** and **.R2**. Recognition of these lines can be prevented by the **-R** option. When a **.R1** line is recognized any accumulated references are flushed out. Neither **.R1** nor **.R2** lines, nor anything between them is output.

Commands are separated by newlines or ;s. # 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 " extends to the next " that is not followed by another ". If there is no such " the word extends to the end of the line. Pairs of " in a word beginning with " collapse to a single ". Neither # nor ; are recognized inside "s. A line can be continued by ending it with \; this works everywhere except after a #.

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; *filename* 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 last name by *string2*, and from anything else (such as a **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 **.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...* *string...* are definite or indefinite articles, and should be ignored at the beginning of **T** fields when sorting. Initially, **the**, **a** and **an** are recognized as articles.

**bibliography** *filename...* Write out all the references contained in the bibliographic databases *filename...* This command should come last in a **.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 behaviour is

```
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** *filename...*

Search the bibliographic databases *filename...* For each *filename* if an index *filename.i* created by **indxib**(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 the **Label expressions** subsection 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 behaviour, so the negative version of this command is more useful. **ref er** 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* Control use of **et al** in the evaluation of @ expressions in label expressions. If the number of authors needed to make the author sequence unambiguous is *u* 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 behaviour is
- et-al " et al" 2 3**
- include** *filename* Include *filename* and interpret the contents as commands.
- join-authors** *string1 string2 string3* This says how authors should be joined together. 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 behaviour; 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 behaviour; 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. It is usually a good idea to give this command

- unless you are using superscripted numbers as labels.
- reverse\*** *string* Reverse the fields whose names are in *string*. Each field name can be followed by a number which says how many such fields should be reversed. If no number is given for a field, all such fields will be reversed.
- 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. (The **Label expressions** subsection 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 will have 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*

*field n*

The *n*-th part of *field*. If *n* is omitted, it defaults to 1.

*'string'*

The characters in *string* literally.

- @ All the authors joined as specified by the **join-authors** command. The whole of each author's name will be used. However, if the references are sorted by author (that is the sort specification starts with **A**), then authors last names 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. The use of only the last name for the *i*-th author of some reference is considered to be ambiguous if there is some other reference, such that the first *i* - 1 authors of the references are the same, the *i*-th authors are not the same, but the *i*-th authors last names 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 purpose will have the same representation.

**%n**

**%a**

**%A**

**%i**

- %l** 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.

*exprn*

*expr-n*

The first ( ) or last (-) *n* upper or lower case letters or digits of *expr*. Troff special characters (such as **\('a**) count as a single letter. Accent strings are retained but do not count towards 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 last name is first.

***expr.a***

*expr* with first names abbreviated. Note that 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.***

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 last name 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 concatenation 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. Note that it is permissible for the first part to be empty; this maybe 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 number register **[P** is set to 1 if the **P** field contains a range of pages. The **[T**, **[A** and **[O** number registers are set to 1 according as the **T**,

**A** and **O** fields end with one of the characters **.?!.** The **[E** number 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 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.

## FILES

**/usr/dict/papers/Ind** Default database.

**file.i** Index files.

**refer** uses temporary files. See the **gr off(1)** man page for details where such files are created.

## ENVIRONMENT

**REFER** If set, overrides the default database.

## SEE ALSO

**indxbib(1)**, **lookbib(1)**, **lkbib(1)**

## BUGS

In label expressions, **<>** expressions are ignored inside **.char** expressions.

## 6.8. gsoelim

### NAME

soelim – interpret .so requests in groff input

### SYNOPSIS

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

It is possible to have whitespace between the **-I** command-line option and its parameter.

### DESCRIPTION

**soelim** reads *files* and replaces lines of the form

**.so** *file*

by the contents of *file*. It is useful if files included with **.so** need to be preprocessed. Normally, **soelim** should be invoked with the **-s** option of **groff**.

To embed ‘\’ in the file name, write ‘\\’ or ‘\e’. To embed a space, write ‘\ ’. Any other escape sequence in *file* makes **soelim** ignore the whole line.

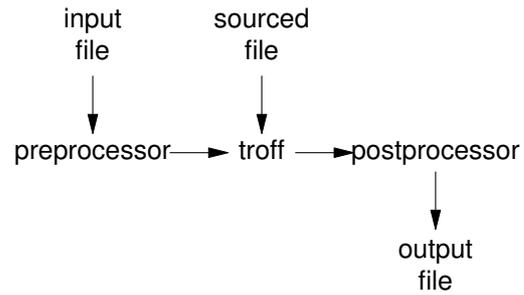
Note that there must be no whitespace between the leading dot and the two characters ‘s’ and ‘o’. Otherwise, only **groff** interprets the **.so** request (and **soelim** ignores it).

### OPTIONS

- C** Recognize **.so** even when followed by a character other than space or newline.
- I***dir* This option may be used to add a directory to the search path for files (both those on the command line and those named in **.so** requests). The search path is initialized with the current directory. This option may be specified more than once; the directories are then searched in the order specified (but before the current directory). If you want to make the current directory be read before other directories, add **-I.** at the appropriate place.  
No directory search is performed for files with an absolute file name.
- r** Do not add **.lf** requests (for general use, with non-groff files).
- t** Don’t emit **.lf** requests but TeX comment lines (starting with ‘%’) giving the current file and line number.
- v** Print the version number.

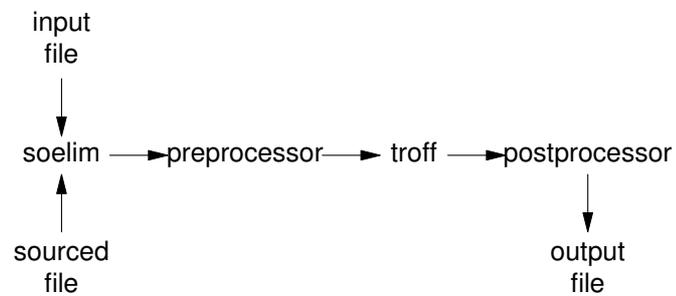
### USAGE

The normal processing sequence of groff is this:



That is, files sourced with **.so** are normally read *only* by **troff** (the actual formatter). **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 file passes through the preprocessor. This is handled by **soelim**:



**SEE ALSO**  
**groff(1)**

## 6.9. preconv

### NAME

preconv – convert encoding of input files to something GNU troff understands

### SYNOPSIS

**preconv** [**-dr**] [**-encoding**] [*files ...*]

**preconv** **-h** | **--help**

**preconv** **-v** | **--version**

It is possible to have whitespace between the **-e** command-line option and its parameter.

### DESCRIPTION

**preconv** reads *files* and converts its encoding(s) to a form GNU **troff**(1) can process, sending the data to standard output. Currently, this means ASCII characters and `[\uXXXX]` entities, where `XXXX` is a hexadecimal number with four to six digits, representing a Unicode input code. Normally, **preconv** should be invoked with the **-k** and **-K** options of **groff**.

### OPTIONS

**-d** Emit debugging messages to standard error (mainly the used encoding).

**-Dencoding**

Specify default encoding if everything fails (see below).

**-encoding**

Specify input encoding explicitly, overriding all other methods. This corresponds to **groff**'s **-Kencoding** option. Without this switch, **preconv** uses the algorithm described below to select the input encoding.

**--help**

**-h** Print help message.

**-r** Do not add `.lf` requests.

**--version**

**-v** Print version number.

### USAGE

**preconv** tries to find the input encoding with the following algorithm.

1. If the input encoding has been explicitly specified with option **-e**, use it.
2. Otherwise, check whether the input starts with a *Byte Order Mark* (BOM, see below). If found, use it.
3. Otherwise, check whether there is a known *coding tag* (see below) in either the first or second input line. If found, use it.
4. Finally, if **uchardet** library (an encoding detector library available on most major distributions) is available on the system, use it to try to detect the encoding of the file.
5. If everything fails, use a default encoding as given with option **-D**, by the current locale, or `'latin1'` if the locale is set to `'C'`, `'POSIX'`, or empty (in that order).

Note that the **groff** program supports a **GROFF\_ENCODING** environment variable which is eventually expanded to option **-k**.

### Byte Order Mark

The Unicode Standard defines character U+FEFF as the Byte Order Mark (BOM). On the other hand, value U+FFFE is guaranteed not be a Unicode character at all. This allows detection of the byte order within the data stream (either big-endian or little-endian), and the MIME encodings 'UTF-16' and 'UTF-32' mandate that the data stream starts with U+FEFF. Similarly, the data stream encoded as 'UTF-8' might start with a BOM (to ease the conversion from and to UTF-16 and UTF-32). In all cases, the byte order mark is *not* part of the data but part of the encoding protocol; in other words, **preconv**'s output doesn't contain it.

Note that U+FEFF not at the start of the input data actually is emitted; it has then the meaning of a 'zero width no-break space' character – something not needed normally in **groff**.

### Coding Tags

Editors which 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 heuristic algorithms for data which represents a greater amount of a natural language, it is still just a guess. Additionally, all algorithms fail easily for input which is either too short or doesn't represent a natural language.

For these reasons, **preconv** supports the coding tag convention (with some restrictions) as used by **GNU Emacs** and **XEmacs** (and probably other programs too).

Coding tags in **GNU Emacs** and **XEmacs** are stored in so-called *File Variables*. **preconv** recognizes the following syntax form which must be put into a troff comment in the first or second line.

```
-*– tag1: value1; tag2: value2; ... –*–
```

The only relevant tag for **preconv** is 'coding' which can take the values listed below. Here an example line which tells **Emacs** to edit a file in troff mode, and to use latin2 as its encoding.

```
.\ " –*– mode: troff; coding: latin-2 –*–
```

The following list gives all MIME coding tags (either lowercase or uppercase) supported by **preconv**; this list is hard-coded in the source.

```
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 hard-coded list of other tags is recognized which eventually map to values 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

Those tags are taken from **GNU Emacs** and **XEmacs**, together with some aliases. Trailing '-dos', '-unix', and '-mac' suffixes of coding tags (which give the end-of-line convention used in the file) are stripped off before the comparison with the above tags happens.

### Iconv Issues

**preconv** by itself only supports three encodings: latin-1, cp1047, and UTF-8; all other encodings are passed to the **iconv** library functions. At compile time it is searched and checked for a valid **iconv** implementation; a call to 'preconv --version' shows whether **iconv** is used.

### BUGS

**preconv** doesn't support *local variable lists* yet. This is a different syntax form to specify local variables at the end of a file.

### SEE ALSO

**groff(1)**  
the **GNU Emacs** and **XEmacs** info pages

## 7. Output Devices

### 7.1. Special Characters

See [Font Files](#).

### 7.2. `grotty`

The postprocessor `grotty` translates the output from GNU `troff` into a form suitable for typewriter-like devices. It is fully documented on its manual page, *grotty(1)*.

#### 7.2.1. Invoking `grotty`

The postprocessor `grotty` accepts the following command-line options:

- `-b` Do not overstrike bold glyphs. Ignored if `-c` isn't used.
- `-B` Do not underline bold-italic glyphs. Ignored if `-c` isn't used.
- `-c` Use overprint and disable colours for printing on legacy Teletype printers (see below).
- `-d` Do not render lines (that is, ignore all `\D` escapes).
- `-f` Use form feed control characters in the output.
- `-Fdir` Put the directory `dir/devname` in front of the search path for the font and device description files, given the target device `name`.
- `-h` Use horizontal tabs for sequences of 8 space characters.
- `-i` Request italic glyphs from the terminal. Ignored if `-c` is active.
- `-o` Do not overstrike.
- `-r` Highlight italic glyphs. Ignored if `-c` is active.
- `-u` Do not underline italic glyphs. Ignored if `-c` isn't used.
- `-U` Do not overstrike bold-italic glyphs. Ignored if `-c` isn't used.
- `-v` Print the version number.

The `-c` mode for TTY output devices means that underlining is done by emitting sequences of `'_'` and `^H` (the backspace character) before the actual character. Literally, this is printing an underline character, then moving the caret back one character position, and printing the actual character at the same position as the underline character (similar to a typewriter). Usually, a modern terminal can't interpret this (and the original Teletype machines for which this sequence was appropriate are no longer in use). You need a pager program like `less` that translates this into ISO 6429 SGR sequences to control terminals.

### 7.3. `grops`

The postprocessor `grops` translates the output from GNU `troff` into a form suitable for Adobe `POSTSCRIPT` devices. It is fully documented on its manual page, *grops(1)*.

### 7.3.1. Invoking `grops`

The postprocessor `grops` accepts the following command-line options:

- `-bflags` Use backward compatibility settings given by *flags* as documented in the *grops(1)* manual page. Overrides the command `broken` in the `DESC` file.
- `-cn` Print *n* copies of each page.
- `-Fdir` Put the directory *dir/devname* in front of the search path for the font, prologue and device description files, given the target device *name*, usually **ps**.
- `-g` Tell the printer to guess the page length. Useful for printing vertically centered pages when the paper dimensions are determined at print time.
- `-Ipath ...`  
Consider the directory *path* for searching included files specified with relative paths. The current directory is searched as fallback.
- `-l` Use landscape orientation.
- `-m` Use manual feed.
- `-ppapersize`  
Set the page dimensions. Overrides the commands `papersize`, `paperlength`, and `paperwidth` in the `DESC` file. See the *roff\_font(5)* manual page for details.
- `-Pprologue`  
Use the *prologue* in the font path as the prologue instead of the default `prologue`. Overrides the environment variable `GROPS_PROLOGUE`.
- `-wn` Set the line thickness to *n*/1000 em. Overrides the default value *n* = 40.
- `-v` Print the version number.

### 7.3.2. Embedding PostScript

The escape sequence

```
\X'ps: import file llx lly urx ury width [height]'
```

places a rectangle of the specified *width* containing the `POSTSCRIPT` drawing from file *file* bound by the box from *llx lly* to *urx ury* (in `POSTSCRIPT` coordinates) at the insertion point. If *height* is not specified, the embedded drawing is scaled proportionally.

See [Miscellaneous](#), for the `psbb` request, which automatically generates the bounding box.

This escape sequence is used internally by the macro `PSPIC` (see the *groff\_tmac(5)* manual page).

## 7.4. `gropdf`

The postprocessor `gropdf` translates the output from GNU `troff` into a form suitable for Adobe PDF devices. It is fully documented on its manual page, *gropdf(1)*.

### 7.4.1. Invoking `gropdf`

The postprocessor `gropdf` accepts the following command-line options:

- d        Produce uncompressed PDFs that include debugging comments.
- e        This forces `gropdf` to embed all used fonts in the PDF, even if they are one of the 14 base Adobe fonts.
- F*dir*    Put the directory *dir/devname* in front of the search path for the font, prologue and device description files, given the target device *name*, usually **pdf**.
- y*foundry*  
      This forces the use of a different font foundry.
- l        Use landscape orientation.
- ppapersize  
      Set the page dimensions. Overrides the commands `papersize`, `paperlength`, and `paperwidth` in the `DESC` file. See the *gropdf* manual page for details.
- v        Print the version number.
- s        Append a comment line to end of PDF showing statistics, i.e. number of pages in document. Ghostscript's *ps2pdf(1)* complains about this line if it is included, but works anyway.
- u*filename*  
      `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 that contain ligatures. You can include your own CMap by specifying a *filename* or have no CMap at all by omitting the *filename*.

### 7.4.2. Embedding PDF

The escape sequence

```
\X'pdf: pdfpic file alignment width [height] [linelength]'
```

places a rectangle of the specified *width* containing the PDF drawing 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 *linelength* 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.

## 7.5. `grodvi`

The postprocessor `grodvi` translates the output from GNU `troff` into the **DVI** output format compatible with the **TEX** document preparation system. It is fully documented on its manual page, *grodvi(1)*.

### 7.5.1. Invoking `grodvi`

The postprocessor `grodvi` accepts the following command-line options:

- `-d` Do not use **tpic** specials to implement drawing commands.
- `-Fdir` Put the directory `dir/devname` in front of the search path for the font and device description files, given the target device *name*, usually **dvi**.
- `-l` Use landscape orientation.
- `-ppapersize` Set the page dimensions. Overrides the commands `papersize`, `paperlength`, and `paperwidth` in the `DESC` file. See *roff\_font(5)* manual page for details.
- `-v` Print the version number.
- `-wn` Set the line thickness to  $n/1000$  em. Overrides the default value  $n = 40$ .

## 7.6. `grolj4`

The postprocessor `grolj4` translates the output from GNU `troff` into the **PCL5** output format suitable for printing on a **HP LaserJet 4** printer. It is fully documented on its manual page, *grolj4(1)*.

### 7.6.1. Invoking `grolj4`

The postprocessor `grolj4` accepts the following command-line options:

- `-cn` Print  $n$  copies of each page.
- `-Fdir` Put the directory `dir/devname` in front of the search path for the font and device description files, given the target device *name*, usually **lj4**.
- `-l` Use landscape orientation.
- `-psize` Set the page dimensions. Valid values for *size* are: `letter`, `legal`, `executive`, `a4`, `com10`, `monarch`, `c5`, `b5`, `d1`.
- `-v` Print the version number.
- `-wn` Set the line thickness to  $n/1000$  em. Overrides the default value  $n = 40$ .

The special drawing command `\D'R dh dv'` draws a horizontal rectangle from the current position to the position at offset  $(dh, dv)$ .

## 7.7. `grolbp`

The postprocessor `grolbp` translates the output from GNU `troff` into the **LBP** output format suitable for printing on **Canon CAPSL** printers. It is fully documented on its manual page, *grolbp(1)*.

### 7.7.1. Invoking `grolbp`

The postprocessor `grolbp` accepts the following command-line options:

- `-cn`        Print *n* copies of each page.
- `-Fdir`     Put the directory *dir/devname* in front of the search path for the font, prologue and device description files, given the target device *name*, usually **lbp**.
- `-l`         Use landscape orientation.
- `-orientation`  
          Use the *orientation* specified: `portrait` or `landscape`.
- `-ppapersize`  
          Set the page dimensions. See *g roff\_font(5)* manual page for details.
- `-wn`        Set the line thickness to *n*/1000 em. Overrides the default value *n* = 40.
- `-v`         Print the version number.
- `-h`         Print command-line help.

## 7.8. `grohtml`

The `grohtml` front end (which consists of a preprocessor, `pre-grohtml`, and a device driver, `post-grohtml`) translates the output of GNU `troff` to HTML. Users should always invoke `grohtml` via the `groff` command with a `\-Thtml` option. If no files are given, `grohtml` will read the standard input. A filename of `-` will also cause `grohtml` to read the standard input. HTML output is written to the standard output. When `grohtml` is run by `groff`, options can be passed to `grohtml` using `groff`'s `-P` option.

`grohtml` invokes `groff` twice. In the first pass, pictures, equations, and tables are rendered using the `ps` device, and in the second pass HTML output is generated by the `html` device.

`grohtml` always writes output in UTF-8 encoding and has built-in entities for all non-composite unicode characters. In spite of this, `groff` may issue warnings about unknown special characters if they can't be found during the first pass. Such warnings can be safely ignored unless the special characters appear inside a table or equation, in which case glyphs for these characters must be defined for the `ps` device as well.

This output device is fully documented on its manual page, *grohtml(1)*.

### 7.8.1. Invoking `grohtml`

The postprocessor `grohtml` accepts the following command-line options:

- `-abits`    Use this number of *bits* (= 1, 2 or 4) for text antialiasing. Default: *bits* = 4.
- `-a0`        Do not use text antialiasing.
- `-b`         Use white background.
- `-Ddir`     Store rendered images in the directory *dir*.

- `-Fdir` Put the directory *dir/devname* in front of the search path for the font, prologue and device description files, given the target device *name*, usually **html**.
- `-gbits` Use this number of *bits* (= 1, 2 or 4) for antialiasing of drawings. Default: *bits* = 4.
- `-g0` Do not use antialiasing for drawings.
- `-h` Use the **B** element for section headings.
- `-iresolution`  
Use the *resolution* for rendered images. Default: *resolution* = 100 dpi.
- `-Istem` Set the images' *stem name*. Default: *stem* = *grohtml-XXX* (*XXX* is the process ID).
- `-jstem` Place each section in a separate file called *stem-n.html* (where *n* is a generated section number).
- `-l` Do not generate the table of contents.
- `-n` Generate simple fragment identifiers.
- `-ooffset`  
Use vertical padding *offset* for images.
- `-p` Display the page rendering progress to `stderr`.
- `-r` Do not use horizontal rules to separate headers and footers.
- `-ssize` Set the base font size, to be modified using the elements **BIG** and **SMALL**.
- `-Slevel` Generate separate files for sections at level *level*.
- `-v` Print the version number.
- `-V` Generate a validator button at the bottom.
- `-y` Generate a signature of `groff` after the validator button, if any.

### 7.8.2. `grohtml` specific registers and strings

`\n[ps4html]`

`\*[www-image-template]`

The registers `ps4html` and `www-image-template` are defined by the `pre-grohtml` preprocessor. `pre-grohtml` reads in the `troff` input, marks up the inline equations and passes the result firstly to

```
troff -Tps -rps4html=1 -dwww-image-template=template
```

and secondly to

```
troff -Thtml
```

or

```
troff -Txhtml
```

The `POSTSCRIPT` device is used to create all the image files (for `-Thtml`; if `-Txhtml` is

used, all equations are passed to `geqn` to produce MathML, and the register `ps4html` enables the macro sets to ignore floating keeps, footers, and headings.

The register `www-image-template` is set to the user specified template name or the default name.

## 7.9. `gxditview`

### 7.9.1. Invoking `gxditview`

## 8. File formats

All files read and written by `gtroff` are text files. The following two sections describe their format.

### 8.1. `gtroff` Output

This section describes the intermediate output format of GNU `troff`. This output is produced by a run of `gtroff` before it is fed into a device postprocessor program.

As `groff` is a wrapper program around `gtroff` that automatically calls a postprocessor, this output does not show up normally. This is why it is called *intermediate*. `groff` provides the option `-Z` to inhibit postprocessing, such that the produced intermediate output is sent to standard output just like calling `gtroff` manually.

Here, the term *troff output* describes what is output by `gtroff`, while *intermediate output* refers to the language that is accepted by the parser that prepares this output for the postprocessors. This parser is smarter on whitespace and implements obsolete elements for compatibility, otherwise both formats are the same.<sup>28</sup>

The main purpose of the intermediate output concept is to facilitate the development of postprocessors by providing a common programming interface for all devices. It has a language of its own that is completely different from the `gtroff` language. While the `gtroff` language is a high-level programming language for text processing, the intermediate output language is a kind of low-level assembler language by specifying all positions on the page for writing and drawing.

The intermediate output produced by `gtroff` is fairly readable, while output from AT&T `troff` is rather hard to understand because of strange habits that are still supported, but not used any longer by `gtroff`.

#### 8.1.1. Language Concepts

During the run of `gtroff`, the input data is cracked down to the information on what has to be printed at what position on the intended device. So the language of the intermediate output format can be quite small. Its only elements are commands with and without arguments. In this section, the term *command* always refers to the intermediate output language, and never to the `gtroff` language used for document formatting. There are commands for positioning and text writing, for drawing, and for device controlling.

##### 8.1.1.1. Separation

AT&T `troff` output has strange requirements on whitespace. The `gtroff` output parser, however, is smart about whitespace by making it maximally optional. The whitespace characters, i.e., the tab, space, and newline characters, always have a syntactical meaning. They are never printable because spacing within the output is always done by positioning commands.

Any sequence of space or tab characters is treated as a single *syntactical space*. It separates commands and arguments, but is only required when there would occur a clashing between the command code and the arguments without the space. Most often, this happens when variable-length command names, arguments, argument lists, or command

---

<sup>28</sup> The parser and postprocessor for intermediate output can be found in the file `groff-source-dir/src/libs/libdriver/input.cpp`.

clusters meet. Commands and arguments with a known, fixed length need not be separated by syntactical space.

A line break is a syntactical element, too. Every command argument can be followed by whitespace, a comment, or a newline character. Thus *asyntactical line break* is defined to consist of optional syntactical space that is optionally followed by a comment, and a newline character.

The normal commands, those for positioning and text, consist of a single letter taking a fixed number of arguments. For historical reasons, the parser allows stacking of such commands on the same line, but fortunately, in `gtroff`'s intermediate output, every command with at least one argument is followed by a line break, thus providing excellent readability.

The other commands -- those for drawing and device controlling -- have a more complicated structure; some recognize long command names, and some take a variable number of arguments. So all 'D' and 'x' commands were designed to request a syntactical line break after their last argument. Only one command, 'x X', has an argument that can stretch over several lines; all other commands must have all of their arguments on the same line as the command, i.e., the arguments may not be split by a line break.

Empty lines (these are lines containing only space and/or a comment), can occur everywhere. They are just ignored.

#### 8.1.1.2. Argument Units

Some commands take integer arguments that are assumed to represent values in a measurement unit, but the letter for the corresponding scale indicator is not written with the output command arguments. Most commands assume the scale indicator 'u', the basic unit of the device, some use 'z', the scaled point unit of the device, while others, such as the color commands, expect plain integers.

Note that single characters can have the eighth bit set, as can the names of fonts and special characters. The names of characters and fonts can be of arbitrary length. A character that is to be printed is always 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.

#### 8.1.1.3. Document Parts

A correct intermediate output document consists of two parts, the *prologue* and the *body*.

The task of the prologue is to set the general device parameters using three exactly specified commands. `gtroff`'s prologue is guaranteed to consist of the following three lines (in that order):

```
x T device
x res n h v
x init
```

with the arguments set as outlined in [Device Control Commands](#). Note that the parser for the intermediate output format is able to swallow additional whitespace and comments as

well even in the prologue.

The body is the main section for processing the document data. Syntactically, it is a sequence of any commands different from the ones used in the prologue. Processing is terminated as soon as the first 'x stop' command is encountered; the last line of any `gtroff` intermediate output always contains such a command.

Semantically, the body is page oriented. A new page is started by a 'p' command. Positioning, writing, and drawing commands are always done within the current page, so they cannot occur before the first 'p' command. Absolute positioning (by the 'H' and 'V' commands) is done relative to the current page; all other positioning is done relative to the current location within this page.

### 8.1.2. Command Reference

This section describes all intermediate output commands, both from AT&T `troff` as well as the `gtroff` extensions.

#### 8.1.2.1. Comment Command

`#anything<end of line>`

A comment. Ignore any characters from the '#' character up to the next newline character.

This command is the only possibility for commenting in the intermediate output. Each comment can be preceded by arbitrary syntactical space; every command can be terminated by a comment.

#### 8.1.2.2. 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 only necessary when two integer arguments would clash or if the preceding argument ends with a string argument.

`C xxx<whitespace>`

Print a special character named `xxx`. The trailing syntactical space or line break is necessary to allow glyph names of arbitrary length. The glyph is printed at the current print position; the glyph's size is read from the font file. The print position is not changed.

`c g` Print glyph `g` at the current print position;<sup>29</sup> the glyph's size is read from the font file. The print position is not changed.

`f n` Set font to font number `n` (a non-negative integer).

`H n` Move right to the absolute vertical position `n` (a non-negative integer in basic units 'u' relative to left edge of current page).

<sup>29</sup> 'c' is actually a misnomer since it outputs a glyph.

- `h n` Move  $n$  (a non-negative integer) basic units 'u' horizontally to the right. The original UNIX troff manual allows negative values for  $n$  also, but `gtroff` doesn't use this.
- `m color-scheme [component ...]`  
Set the color for text (glyphs), line drawing, and the outline of graphic objects using different color schemes; the analogous command for the filling color of graphic objects is 'DF'. 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 `gtroff`'s escape sequence `\m`. No position changing. These commands are a `gtroff` extension.
- `mc cyan magenta yellow`  
Set color using the CMY color scheme, having the 3 color components *cyan*, *magenta*, and *yellow*.
- `md` Set color to the default color value (black in most cases). No component arguments.
- `mg gray` Set color to the shade of gray given by the argument, an integer between 0 (black) and 65536 (white).
- `mk cyan magenta yellow black`  
Set color using the CMYK color scheme, having the 4 color components *cyan*, *magenta*, *yellow*, and *black*.
- `mr red green blue`  
Set color using the RGB color scheme, having the 3 color components *red*, *green*, and *blue*.
- `N n` Print glyph with index  $n$  (a non-negative integer) of the current font. This command is a `gtroff` extension.
- `n b a` Inform the device about a line break, but no positioning is done by this command. In AT&T `troff`, the integer arguments  $b$  and  $a$  informed about the space before and after the current line to make the intermediate output more human readable without performing any action. In `groff`, they are just ignored, but they must be provided for compatibility reasons.
- `p n` Begin a new page in the outprint. The page number is set to  $n$ . This page is completely independent of pages formerly processed even if those have the same page number. The vertical position on the outprint is automatically set to 0. All positioning, writing, and drawing is always done relative to a page, so a 'p' command must be issued before any of these commands.
- `s n` Set point size to  $n$  scaled points (this is unit 'z'). AT&T `troff` used the unit points ('p') instead. See [Output Language Compatibility](#).
- `t xxx<whitespace>`  
`t xxx dummy-arg<whitespace>`  
Print a word, i.e., a sequence of characters `xxx` representing output glyphs

which names are single characters, terminated by a space character or a line break; an optional second integer argument is ignored (this allows the formatter to generate an even number of arguments). The first glyph should be printed at the current position, the current horizontal position should then be increased by the width of the first glyph, and so on for each glyph. The widths of the glyphs are read from the font file, scaled for the current point size, and rounded to a multiple of the horizontal resolution. Special characters cannot be printed using this command (use the 'C' command for special characters). This command is a `gtroff` extension; it is only used for devices whose `DESC` file contains the `tcommand` keyword (see [DESC File Format](#)).

`u n xxx<whitespace>`

Print word with track kerning. This is the same as the 't' command except that after printing each glyph, the current horizontal position is increased by the sum of the width of that glyph and *n* (an integer in basic units 'u'). This command is a `gtroff` extension; it is only used for devices whose `DESC` file contains the `tcommand` keyword (see [DESC File Format](#)).

`V n` Move down to the absolute vertical position *n* (a non-negative integer in basic units 'u') relative to upper edge of current page.

`v n` Move *n* basic units 'u' down (*n* is a non-negative integer). The original UNIX troff manual allows negative values for *n* also, but `gtroff` doesn't use this.

`w` Informs about a paddable white space to increase readability. The spacing itself must be performed explicitly by a move command.

### 8.1.2.3. Graphics Commands

Each graphics or drawing command in the intermediate output 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.

`gtroff` output follows the classical spacing rules (no space between command and subcommand, all arguments are preceded by a single space character), but the 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 arguments called *h1*, *h2*, ..., *hn* stand for horizontal distances where positive means right, negative left. The arguments called *v1*, *v2*, ..., *vn* 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 similar `gtroff \D` escape sequence. See [Drawing Requests](#).

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 above.

D~ *h1 v1 h2 v2 ... hn vn*<line break>

Draw B-spline from current position to offset  $(h1,v1)$ , then to offset  $(h2,v2)$ , if given, etc. up to  $(hn,vn)$ . This command takes a variable number of argument pairs; the current position is moved to the terminal point of the drawn curve.

Da *h1 v1 h2 v2*<line break>

Draw arc from current position to  $(h1,v1)+(h2,v2)$  with center at  $(h1,v1)$ ; 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 `gtroff` 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 `gtroff` 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 `gtroff`'s escape sequences `\D'F ...'` and `\M` (with no other corresponding graphics commands). No position changing. This command is a `gtroff` 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.

@math{0 @value{LE *n* @value{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 is obsoleted by command 'DFg'.

@math{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.

No position changing. This command is a `gtroff` extension.

DI *h @var{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 *h1 @var{v1 h2 v2 ... hn vn*<line break>

Draw a polygon line from current position to offset (*h1,v1*), from there to offset (*h2,v2*), etc. up to offset (*hn,vn*), 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 actual horizontal position and the even ones to the vertical position. Although this doesn't make sense it is kept for compatibility. @ignore As the polygon is closed, the end of drawing is the starting point, so the position doesn't change. This command is a `agtroff` extension.

DP *h1 @var{v1 h2 v2 ... hn vn*<line break>

Draw a solid polygon in the current fill color rather than an outlined polygon, using the same arguments and positioning as the corresponding 'Dp' command. @ignore No position changing. This command is a `gtroff` 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; if *n*<0 set the line thickness

proportional to the point size (this is the default before the first ‘Dt’ command was specified). For historical reasons, the horizontal position is changed by adding the argument to the actual horizontal position, while the vertical position is not changed. Although this doesn’t make sense it is kept for compatibility. @ignore No position changing. This command is a `gtroff` extension.

#### 8.1.2.4. Device Control Commands

Each device control command starts with the letter ‘x’, followed by a space character (optional or arbitrary space or tab in `gtroff`) 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 new-line character. All characters of the subcommand word but the first are simply ignored. For example, `gtroff` 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 (see [Separation](#)).

`xF name<line break>`

The ‘F’ stands for *Filename*.

Use *name* as the intended name for the current file in error reports. This is useful for remembering the original file name when `gtroff` uses an internal piping mechanism. The input file is not changed by this command. This command is a `gtroff` extension.

`xf n s<line break>`

The ‘f’ stands for *font*.

Mount font position *n* (a non-negative integer) with font named *s* (a text word). See [Font Positions](#).

`xH n<line break>`

The ‘H’ stands for *Height*.

Set glyph height to *n* (a positive integer in scaled points ‘z’). AT&T `troff` uses the unit points (‘p’) instead. See [Output Language Compatibility](#).

`xi<line break>`

The ‘i’ stands for *init*.

Initialize device. This is the third command of the prologue.

`xp<line break>`

The ‘p’ stands for *pause*.

Parsed but ignored. The original UNIX `troff` manual writes pause device, can be restarted

`xr n h v<line break>`

The ‘r’ stands for *resolution*.

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 prologue.

`xS n<line break>`

The ‘S’ stands for *Slant*.

Set slant to  $n$  (an integer in basic units ‘u’).

`xs<line break>`

The ‘s’ stands for *stop*.

Terminates the processing of the current file; issued as the last command of any intermediate troff output.

`xt<line break>`

The ‘t’ stands for *trailer*.

Generate trailer information, if any. In *gtrroff*, this is actually just ignored.

`xT xxx<line break>`

The ‘T’ stands for *Typesetter*.

Set name of device to word  $xxx$ , a sequence of characters ended by the next white space character. The possible device names coincide with those from the `groff -T` option. This is the first command of the prologue.

`xu n<line break>`

The ‘u’ stands for *underline*.

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 `gtrroff` extension.

`xX anything<line break>`

The ‘X’ stands for *X-escape*.

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 `gtrroff` escape sequence `\X`. The line-continuing feature is a `gtrroff` extension.

### 8.1.2.5. Obsolete Command

In AT&T `troff` output, the writing of a single glyph is mostly done by a very strange command that combines a horizontal move and a single character giving the glyph name. It doesn’t have a command code, but is represented by a 3-character argument consisting of exactly 2 digits and a character.

`dd@var{g}`

Move right  $dd$  (exactly two decimal digits) basic units ‘u’, then print glyph  $g$  (represented as a single character).

In `gtroff`, 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 AT&T `troff`, large clusters of these and other commands are used, mostly without spaces; this made such output almost unreadable.

For modern high-resolution devices, this command does not make sense because the width of the glyphs can become much larger than two decimal digits. In `gtroff`, this is only used for the devices `x75`, `x75-12`, `x100`, and `x100-12`. For other devices, the commands `'t'` and `'u'` provide a better functionality.

### 8.1.3. Intermediate Output Examples

This section presents the intermediate output generated from the same input for three different devices. The input is the sentence 'hell world' fed into `gtroff` on the command line.

High-resolution device `ps`

This is the standard output of `gtroff` if no `-T` option is given.

```
shell> echo "hell world" | groff -Z -T ps

x T ps
x res 72000 1 1
x init
p1
x font 5 TR
f5
s10000
V12000
H72000
thell
wh2500
tw
H96620
torld
n12000 0
x trailer
V792000
x stop
```

This output can be fed into `grops` to get its representation as a `POSTSCRIPT` file.

Low-resolution device `latin1`

This is similar to the high-resolution device except that the positioning is done at a minor scale. Some comments (lines starting with `'#'`) were added for clarification; they were not generated by the formatter.

```
shell> echo "hell world" | groff -Z -T latin1

prologue
x T latin1
```

```

x res 240 24 40
x init
begin a new page
p1
font setup
x font 1 R
f1
s10
initial positioning on the page
V40
H0
write text 'hell'
thell
inform about space, and issue a horizontal jump
wh24
write text 'world'
tworld
announce line break, but do nothing because ...
n40 0
... the end of the document has been reached
x trailer
V2640
x stop

```

This output can be fed into `grotty` to get a formatted text document.

#### @acronym{AT&T troff output}

Since a computer monitor has a very low resolution compared to modern printers the intermediate output for the X Window devices can use the jump-and-write command with its 2-digit displacements.

```

shell> echo "hell world" | groff -Z -T X100

x T X100
x res 100 1 1
x init
p1
x font 5 TR
f5
s10
V16
H100
write text with jump-and-write commands
ch07e07103lw06w11o07r05l03dh7
n16 0
x trailer
V1100
x stop

```

This output can be fed into `xditview` or `gxditview` for displaying in X.

Due to the obsolete jump-and-write command, the text clusters in the AT&T

`troff` output are almost unreadable.

### 8.1.4. Output Language Compatibility

The intermediate output language of AT&T `troff` was first documented in the UNIX `troff` manual, with later additions documented in *A Typesetter-independent TROFF*, written by Brian Kernighan.

The `gtroff` intermediate output format is compatible with this specification except for the following features.

- The classical quasi device independence is not yet implemented.
- The old hardware was very different from what we use today. So the `groff` devices are also fundamentally different from the ones in AT&T `troff`. For example, the AT&T `POSTSCRIPT` device is called `post` and has a resolution of only 720 units per inch, suitable for printers 20 years ago, while `groff`'s `ps` device has a resolution of 72000 units per inch. Maybe, by implementing some rescaling mechanism similar to the classical quasi device independence, `groff` could emulate AT&T's `post` device.
- The B-spline command 'D~' is correctly handled by the intermediate output parser, but the drawing routines aren't implemented in some of the postprocessor programs.
- The argument of the commands 's' and 'x H' has the implicit unit scaled point 'z' in `gtroff`, while AT&T `troff` has point ('p'). This isn't an incompatibility but a compatible extension, for both units coincide for all devices without a `sizescale` parameter in the `DESC` file, including all postprocessors from AT&T and `groff`'s text devices. The few `groff` devices with a `sizescale` parameter 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 position changing after the commands 'Dp', 'DP', and 'Dt' is illogical, but as old versions of `gtroff` used this feature it is kept for compatibility reasons.

## 8.2. Font Files

The `gtroff` font format is roughly a superset of the `ditroff` font format (as used in later versions of AT&T `troff` and its descendants). Unlike the `ditroff` font format, there is no associated binary format; all files are text files.<sup>30</sup> The font files for device *name* are stored in a directory *devname*. There are two types of file: a device description file called `DESC` and for each font *f* a font file called *f*.

### 8.2.1. DESC File Format

The `DESC` file can contain the following types of line. Except for the `charset` keyword, which must come last (if at all), the order of the lines is not important. Later entries in the file, however, override previous values.

<sup>30</sup> Plan 9 `troff` has also abandoned the binary format.

`charset` This line and everything following in the file are ignored. It is allowed for the sake of backwards compatibility.

`family` *fam*  
The default font family is *fam*.

`fonts` *n F1 F2 F3 ... Fn*  
Fonts *F1 ... Fn* are mounted in the font positions *m+1, ..., m+n* where *m* is the number of styles. This command may extend over more than one line. A font name of 0 means no font is mounted on the corresponding font position.

`hor` *n* The horizontal resolution is *n* machine units. All horizontal quantities are rounded to be multiples of this value.

`image_generator` *string*  
Needed for `grohtml` only. It specifies the program to generate PNG images from `POSTSCRIPT` input. Under GNU/Linux this is usually `gs` but under other systems (notably `cygwin`) it might be set to another name.

`paperlength` *n*  
The physical vertical dimension of the output medium in machine units. This isn't used by `troff` itself but by output devices. Deprecated. Use `papersize` instead.

`papersize` *string ...*  
Select a paper size. Valid values for *string* are the ISO paper types `A0--A7`, `B0--B7`, `C0--C7`, `D0--D7`, `DL`, and the US paper types `letter`, `legal`, `tabloid`, `ledger`, `statement`, `executive`, `com10`, and `monarch`. Case is not significant for *string* if it holds predefined paper types. Alternatively, *string* can be a file name (e.g. `/etc/papersize`); if the file can be opened, `groff` reads the first line and tests for the above paper sizes. Finally, *string* can be a custom paper size in the format *length,width* (no spaces before and after the comma). Both *length* and *width* must have a unit appended; valid values 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. `papersize` sets both the vertical and horizontal dimension of the output medium.

More than one argument can be specified; `groff` scans from left to right and uses the first valid paper specification.

`paperwidth` *n*  
The physical horizontal dimension of the output medium in machine units. This isn't used by `troff` itself but by output devices. Deprecated. Use `papersize` instead.

`pass_filenames`  
Tell `gtroff` to emit the name of the source file currently being processed. This is achieved by the intermediate output command 'F'. Currently, this is only used by the `grohtml` output device.

`postpro` *program*  
Call *program* as a postprocessor. For example, the line

postpro grodvi

in the file devdvi/DESC makes groff call grodvi if option `-Tdvi` is given (and `-Z` isn't used).

prepro *program*

Call *program* as a preprocessor. Currently, this keyword is used by groff with option `-Thtml` or `-Txhtml` only.

print *program*

Use *program* as a spooler program for printing. If omitted, the `-l` and `-L` options of groff are ignored.

res *n* There are *n* machine units per inch.

sizes *s1 s2 ... sn 0*

This means that the device has fonts at *s1*, *s2*, ... *sn* scaled points. The list of sizes must be terminated by 0 (this is digit zero). Each *si* can also be a range of sizes *m--n*. The list can extend over more than one line.

sizescale *n*

The scale factor for point sizes. By default this has a value of 1. One scaled point is equal to one point/*n*. The arguments to the `unitwidth` and `sizes` commands are given in scaled points. See [Fractional Type Sizes](#), for more information.

styles *S1 S2 ... Sm*

The first *m* font positions are associated with styles *S1 ... Sm*.

tcommand

This means that the postprocessor can handle the 't' and 'u' intermediate output commands.

unicode Indicate that the output device supports the complete Unicode repertoire. Useful only for devices that 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 `charset` section, they either override the default mappings for those particular characters or add new mappings (normally for composite characters).

This is used for `-Tutf8`, `-Thtml`, and `-Txhtml`.

unitwidth *n*

Quantities in the font files are given in machine units for fonts whose point size is *n* scaled points.

unscaled\_charwidths

Make the font handling module always return unscaled character widths. Needed for the `grohtml` device.

use\_charnames\_in\_special

This command indicates that `gtroff` should encode special characters inside special commands. Currently, this is only used by the `grohtml` output device.

See [Postprocessor Access](#).

`vert n` The vertical resolution is  $n$  machine units. All vertical quantities are rounded to be multiples of this value.

The `res`, `unitwidth`, `fonts`, and `sizes` lines are mandatory. Other commands are ignored by `gtroff` but may be used by postprocessors to store arbitrary information about the device in the `DESC` file.

Here a list of obsolete keywords that are recognized by `groff` but completely ignored: `spare1`, `spare2`, `biggestfont`.

### 8.2.2. Font File Format

A *font file*, also (and probably better) called a *font description file*, has two sections. The first section is a sequence of lines each containing a sequence of blank delimited words; the first word in the line is a key, and subsequent words give a value for that key.

`name f` The name of the font is  $f$ .

`spacewidth n`  
The normal width of a space is  $n$ .

`slant n` The glyphs of the font have a slant of  $n$  degrees. (Positive means forward.)

`ligatures lig1 lig2 ... ligm [0]`  
Glyphs  $lig1$ ,  $lig2$ , ...,  $ligm$  are ligatures; possible ligatures are ‘ff’, ‘fi’, ‘fl’, ‘ffi’ and ‘ffl’. For backwards compatibility, the list of ligatures may be terminated with a 0. The list of ligatures may not extend over more than one line.

`special` The font is *special*; this means that when a glyph is requested that is not present in the current font, it is searched for in any special fonts that are mounted.

Other commands are ignored by `gtroff` but may be used by postprocessors to store arbitrary information about the font in the font file.

The first section can contain comments, which start with the ‘#’ character and extend to the end of a line.

The second section contains one or two subsections. It must contain a `charset` subsection and it may also contain a `kernpairs` subsection. These subsections can appear in any order. Each subsection starts with a word on a line by itself.

The word `charset` starts the character set subsection.<sup>31</sup> The `charset` line is followed by a sequence of lines. Each line gives information for one glyph. A line comprises a number of fields separated by blanks or tabs. The format is

*name metrics type code [entity-name] [-- comment]*

*name* identifies the glyph name<sup>32</sup>: If *name* is a single character  $c$  then it corresponds to the `gtroff` input character  $c$ ; if it is of the form ‘\c’ where  $c$  is a single character, then it corresponds to the special character `\[c]`; otherwise it corresponds to the special

<sup>31</sup> This keyword is misnamed since it starts a list of ordered glyphs, not characters.

<sup>32</sup> The distinction between input, characters, and output, glyphs, is not clearly separated in the terminology of `groff`; for example, the `char` request should be called `glyph` since it defines an output entity.

character `\[name]`. If it is exactly two characters *xx* it can be entered as `\(xx)`. Note that single-letter special characters can't be accessed as `\c`; the only exception is `\-`, which is identical to `\[-]`.

`gtroff` supports 8-bit input characters; however some utilities have difficulties with eight-bit characters. For this reason, there is a convention that the entity name `'charn'` is equivalent to the single input character whose code is *n*. For example, `'char163'` would be equivalent to the character with code 163, which is the pounds sterling sign in the ISO Latin-1 character set. You shouldn't use `'charn'` entities in font description files since they are related to input, not output. Otherwise, you get hard-coded connections between input and output encoding, which prevents use of different (input) character sets.

The name `'---`' is special and indicates that the glyph is unnamed; such glyphs can only be used by means of the `\N` escape sequence in `gtroff`.

The *type* field gives the glyph type:

- 1 the glyph has a descender, for example, 'p';
- 2 the glyph has an ascender, for example, 'b';
- 3 the glyph has both an ascender and a descender, for example, 'i'.

The *code* field gives the code that the postprocessor uses to print the glyph. The glyph can also be input to `gtroff` using this code by means of the `\N` escape sequence. *code* can be any integer. If it starts with '0' it is interpreted as octal; if it starts with '0x' or '0X' it is interpreted as hexadecimal. Note, however, that the `\N` escape sequence only accepts a decimal integer.

The *entity-name* field gives an ASCII string identifying the glyph that the postprocessor uses to print the `gtroff` glyph *name*. This field is optional and has been introduced so that the `grohtml` device driver can encode its character set. For example, the glyph `\[Po]` is represented as `'&pound;'` in HTML 4.0.

Anything on the line after the *entity-name* field resp. after `'--'` is ignored.

The *metrics* field has the form:

`width[, height[, depth[, italic-correction [, left-italic-correction[, subscript-correction]]]]]`

There must not be any spaces between these subfields (it has been split here into two lines for better legibility only). Missing subfields are assumed to be 0. The subfields are all decimal integers. Since there is no associated binary format, these values are not required to fit into a variable of type 'char' as they are in `ditroff`. The *width* subfield gives the width of the glyph. The *height* subfield gives the height of the glyph (upwards is positive); if a glyph does not extend above the baseline, it should be given a zero height, rather than a negative height. The *depth* subfield gives the depth of the glyph, that is, the distance from the baseline to the lowest point below the baseline to which the glyph extends (downwards is positive); if a glyph does not extend below the baseline, it should be given a zero depth, rather than a negative depth. The *italic-correction* subfield gives the amount of space that should be added after the glyph when it is immediately to be followed by a glyph from a roman font. The *left-italic-correction* subfield gives the amount of space that should be added before the glyph when it is immediately to be preceded by a glyph from a roman font. The *subscript-correction* gives the amount of space that should be added after a glyph before adding a subscript. This should be less than the italic correction.

A line in the `charset` section can also have the format

*name* "

This indicates that *name* is just another name for the glyph mentioned in the preceding line.

The word `kernpairs` starts the `kernpairs` section. This contains a sequence of lines of the form:

*c1 c2 n*

This means that when glyph *c1* appears next to glyph *c2* the space between them should be increased by *n*. Most entries in the `kernpairs` section have a negative value for *n*.

## **9. Installation**

## 10. Copying This Manual

Version 1.3, 3 November 2008

Copyright © 2000, 2001, 2002, 2007, 2008 Free Software Foundation, Inc. <http://fsf.org/>

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

- PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document *free* in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondly, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of “copyleft”, which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

- APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The “Document”, below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as “you”. You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A “Modified Version” of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A “Secondary Section” is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document’s overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The “Invariant Sections” are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The “Cover Texts” are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A “Transparent” copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not “Transparent” is called “Opaque”.

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The “Title Page” means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, “Title Page” means the text near the most prominent appearance of the work’s title, preceding the beginning of the body of the text.

The “publisher” means any person or entity that distributes copies of the Document to the public.

A section “Entitled XYZ” means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as “Acknowledgements”, “Dedications”, “Endorsements”, or “History”.) To “Preserve the Title” of such a section when you modify the Document means that it remains a section “Entitled XYZ” according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

- VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or non-commercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute.

However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

- **COPYING IN QUANTITY**

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

- **MODIFICATIONS**

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

- Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
- List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

- State on the Title page the name of the publisher of the Modified Version, as the publisher.
- Preserve all the copyright notices of the Document.
- Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
- Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
- Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
- Include an unaltered copy of this License.
- Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
- Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
- For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
- Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
- Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.
- Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.
- Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties---for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

- **COMBINING DOCUMENTS**

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled “History” in the various original documents, forming one section Entitled “History”; likewise combine any sections Entitled “Acknowledgements”, and any sections Entitled “Dedications”. You must delete all sections Entitled “Endorsements.”

- **COLLECTIONS OF DOCUMENTS**

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

- **AGGREGATION WITH INDEPENDENT WORKS**

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an “aggregate” if the copyright resulting from the compilation is not used to limit the legal rights of the compilation’s users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document’s Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must

appear on printed covers that bracket the whole aggregate.

- TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled “Acknowledgements”, “Dedications”, or “History”, the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

- TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

- FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See <http://www.gnu.org/copyleft/>.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License “or any later version” applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy’s public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

- RELICENSING

“Massive Multiauthor Collaboration Site” (or “MMC Site”) means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A “Massive Multiauthor Collaboration” (or “MMC”) contained in the site means any set of copyrightable works thus published on the MMC site.

“CC-BY-SA” means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

“Incorporate” means to publish or republish a Document, in whole or in part, as part of another Document.

An MMC is “eligible for relicensing” if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

Copyright (C) *year your name*. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled “GNU Free Documentation License”.

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the “with...Texts.” line with this:

with the Invariant Sections being *list their titles*, with the Front-Cover Texts being *list*, and with the Back-Cover Texts being *list*.

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

```
@samp{^E}=1
c1n:@email{=1
@finalout=1
man:hw=1
@bye=1
@samp{A}=1
@samp{+}=1
@contents=1
@documentlanguage=1
@heading=1
```

```
@top=1
man:mso=1
@titlepage=1
@strong{Caution:}=1
@setchapternewpage=1
@ftable=1
@documentencoding=1
@samp{use_charnames_in_special}=1
@smallbook=1
@enddots{}=1
cln:@c{=1
@vskip=1
@setfilename=1
@footnotestyle=1
man:nr=1
cln:@acronym{=1
cln:@footnote{=1
@samp{u}=1
@deffnx=2
man:.de FONT =2
cln:@value{=2
@ifnottex=2
@enumerate=2
@page=2
@deffn=2
man:SM=2
@smallexample=2
cln:@math{=2
@stindex=3
@iftex=3
@quotation=3
@endDefesc=4
man:ad=4
man:.de Tp =4
@display=5
cln:@var{=5
man:FONT=7
@ifnotinfo=7
@ifinfo=7
@tindex=11
@opindex=21
@end=38
@noindent=136
```

## B Request Index

---

ab, [219](#)  
 ad, [125](#)  
 af, [121](#)  
 aln, [120](#)  
 als, [177](#)  
 am, [184](#)  
 am1, [184](#)  
 ami, [184](#)  
 ami1, [184](#)  
 as, [176](#)  
 as1, [176](#)  
 asciify, [205](#)

- a -

backtrace, [220](#)  
 bd, [166](#)  
 blm, [199](#)  
 box, [202](#)  
 boxa, [202](#)  
 bp, [150](#)  
 br, [124](#)  
 break, [181](#)  
 brp, [126](#)

- b -

c2, [140](#)  
 cc, [140](#)  
 ce, [127](#)  
 cf, [210](#)  
 cflags, [160](#)  
 ch, [197](#)  
 char, [162](#)  
 chop, [177](#)  
 class, [163](#)  
 close, [213](#)  
 color, [208](#)  
 composite, [159](#)  
 continue, [182](#)  
 cp, [224](#)  
 cs, [166](#)  
 cu, [165](#)

- c -

da, [202](#)  
 de, [182](#)  
 de1, [182](#)  
 defcolor, [208](#)

- d -

dei, [182](#)  
 dei1, [182](#)  
 device, [214](#)  
 devicem, [214](#)  
 di, [202](#)  
 do, [224](#)  
 ds, [173](#)  
 ds1, [173](#)  
 dt, [198](#)

- e -

ec, [140](#)  
 ecr, [140](#)  
 ecs, [140](#)  
 el, [180](#)  
 em, [200](#)  
 eo, [140](#)  
 ev, [206](#)  
 evc, [206](#)  
 ex, [219](#)

- f -

fam, [154](#)  
 fc, [139](#)  
 fchar, [162](#)  
 fcolor, [209](#)  
 fj, [124](#)  
 fl, [220](#)  
 fp, [155](#)  
 fschar, [162](#)  
 fspecial, [164](#)  
 ft, [152](#), [156](#)  
 ftr, [153](#)  
 fzoom, [153](#)

- g -

gcolor, [209](#)

- h -

hc, [131](#)  
 hcode, [132](#)  
 hla, [133](#)  
 hlm, [130](#)  
 hpf, [131](#)  
 hpfa, [131](#)  
 hpfcodes, [131](#)  
 hw, [130](#)  
 hy, [128](#)  
 hym, [132](#)

hys, 133

- i -

ie, 180

if, 180

ig, 117

in, 146

it, 198

itc, 198

- k -

kern, 167

- l -

lc, 139

length, 176

lf, 219

lg, 166

linetabs, 138

ll, 147

ls, 134

lsm, 199

lt, 149

- m -

mc, 216

mk, 187

mso, 210

- n -

na, 126

ne, 150

nf, 124

nh, 130

nm, 214

nn, 216

nop, 180

nr, 118, 119, 120

nroff, 144

ns, 135

nx, 211

- o -

open, 213

opena, 213

os, 151

output, 205

- p -

pc, 150

pev, 220

pi, 212

pl, 148

pm, 220

pn, 150

pnr, 220

po, 145

ps, 169

psbb, 217

pso, 210

ptr, 220

pvs, 171

- r -

rchar, 163

rd, 211

return, 184

rfschar, 163

rj, 128

rm, 177

rn, 177

rnn, 119

rr, 119

rs, 135

rt, 187

- s -

schar, 162

shc, 133

shift, 185

sizes, 170

so, 210

sp, 133

special, 164

spreadwarn, 221

ss, 126

sty, 154

substring, 176

sv, 151

sy, 212

- t -

ta, 136

tc, 138

tj, 146

tkf, 167

tl, 149

tm, 219

tm1, 219

tmc, 219

tr, 142

trf, 210

trin, 142

trnt, 143

troff, [144](#)

- u -

uf, [165](#)

ul, [165](#)

unformat, [205](#)

- v -

vpt, [195](#)

vs, [171](#)

- w -

warn, [221](#)

warnscale, [221](#)

wh, [195](#)

while, [180](#)

write, [213](#)

writec, [213](#)

writem, [213](#)

## C Escape Index

---

\			- d -
, 189			
, 168		\D, 192	
\\!, 204		\\d, 188	
\\#, 117			- e -
\\\$, 185			
\\\$*, 185		\\e, 141	
\\\$0, 186		\\E, 141	
\\\$@, 185			- f -
\\\$^, 186			
\\%, 130		\\F, 154	
\\&, 168		\\f, 152, 156	
\\', 160			- g -
\\), 169			
\\*, 173		\\g, 122	
\\-, 160			- h -
\\., 141			
\\0, 189		\\H, 164	
\\:, 130		\\h, 189	
\\?, 204			- k -
\\{, 180			
\\}, 180		\\k, 190	
\\^, 189		%key	
\\_, 160		, 116	
\\', 160		\\, 141	
\\ , 189		\\, 158	
	- < -		
\\<colon>, 130			- l -
	- .XX -		
\\ %key, 116		\\L, 191	
\\ %key, 141		\\l, 191	
\\ %key, 158			- m -
%key, \\, 116			
	- a -	\\M, 209	
\\a, 138		\\m, 209	
\\A, 111			- n -
	- b -	\\N, 160	
\\b, 195		\\n, 120, 120	
\\B, 110			- o -
	- c -	\\O, 207	
\\C, 159		\\o, 190	
\\c, 148			

- p -

`\p`, 126

- r -

`\r`, 188

`\R`, 118, 119

`\RET`, 148

- s -

`\s`, 169

`\S`, 165

`\SP`, 189

- t -

`\t`, 136

- u -

`\u`, 188

- v -

`\V`, 214

`\v`, 188

- w -

`\w`, 189

- x -

`\x`, 135

`\X`, 214

- y -

`\Y`, 214

- z -

`\Z`, 190

`\z`, 190



FPS [ms], <a href="#">33</a>			
FVS [ms], <a href="#">33</a>			- r -
		rsb, <a href="#">189</a>	
		rst, <a href="#">189</a>	
	- g -		
GROWPS [ms], <a href="#">31</a>			- s -
GS [ms], <a href="#">51</a>		sb, <a href="#">189</a>	
		seconds, <a href="#">123</a>	
	- h -	skw, <a href="#">189</a>	
HM [ms], <a href="#">30</a>		slimit, <a href="#">221</a>	
HORPHANS [ms], <a href="#">32</a>		ssc, <a href="#">189</a>	
hours, <a href="#">123</a>		st, <a href="#">189</a>	
hp, <a href="#">190</a>		systat, <a href="#">212</a>	
HY [ms], <a href="#">31</a>			- u -
	- l -	urx, <a href="#">217</a>	
LL [ms], <a href="#">30</a>		ury, <a href="#">217</a>	
llx, <a href="#">217</a>			- v -
lly, <a href="#">217</a>			
ln, <a href="#">123</a>		VS [ms], <a href="#">31</a>	
lsn, <a href="#">199</a>			
lss, <a href="#">199</a>			- y -
LT [ms], <a href="#">30</a>		year, <a href="#">123</a>	
	- m -	yr, <a href="#">123</a>	
MINGW [ms], <a href="#">33</a> , <a href="#">53</a>			
minutes, <a href="#">123</a>			
mo, <a href="#">123</a>			
	- n -		
nl, <a href="#">151</a>			
	- o -		
opmaxx, <a href="#">207</a>			
opmaxy, <a href="#">207</a>			
opminx, <a href="#">207</a>			
opminy, <a href="#">207</a>			
	- p -		
PD [ms], <a href="#">32</a>			
PI [ms], <a href="#">32</a>			
PO [ms], <a href="#">30</a>			
PORPHANS [ms], <a href="#">32</a>			
PS [ms], <a href="#">31</a>			
ps4html [grohtml], <a href="#">293</a>			
PSINCR [ms], <a href="#">31</a>			
	- q -		
QI [ms], <a href="#">32</a>			

## F Macro Index

---

1C [ms], 47		Ds	
2C [ms], 47		[man], 27	
		[ms], 44	
	- [ -	DT [man], 25	
[ [ms], 45			- e -
	- ] -	EE [man], 27	
		EF [ms], 46	
] [ms], 45		EH [ms], 46	
	- a -	EN [ms], 45	
		EQ [ms], 45	
		EX [man], 27	
AB [ms], 34			- f -
AE [ms], 35			
AI [ms], 34			
AM [ms], 49, 52		FE [ms], 45	
AT [man], 25		FS [ms], 45	
AU [ms], 34			
	- b -		- g -
B		G [man], 27	
[man], 25		GL [man], 27	
[ms], 38			- h -
B1 [ms], 44			
B2 [ms], 44		HB [man], 27	
BD [ms], 43		HD [ms], 46	
BI		HP [man], 24	
[man], 24			- i -
[ms], 39			
BR [man], 25			
BT		I	
[man], 27		[man], 25	
[ms], 47		[ms], 39	
BX [ms], 39		IB [man], 24	
		ID [ms], 43	
	- c -	IP	
		[man], 23	
		[ms], 39	
CD [ms], 43		IR [man], 25	
CT [man], 27		IX [ms], 53	
CW			- k -
[man], 27			
[ms], 39, 52			
	- d -	KE [ms], 44, 44	
		KF [ms], 44	
DA [ms], 34		KS [ms], 44	
De			- l -
[man], 27			
[ms], 44			
DE [ms], 43, 43, 43, 43, 43		LD [ms], 43	
DS [ms], 43, 43, 43, 43, 43, 52		LG [ms], 39	

LP		RP [ms], 34	
[man], 23		RS	
[ms], 36		[man], 24	
		[ms], 42	
	- m -		- s -
MC [ms], 47		SB [man], 24	
MS [man], 28		SH	
		[man], 23	
	- n -	[ms], 38	
ND [ms], 34		SM	
NE [man], 28		[man], 24	
NH [ms], 37		[ms], 39	
NL [ms], 39		SS [man], 23	
NT [man], 28			- t -
	- o -	TA [ms], 43	
OF [ms], 46		TB [man], 27	
OH [ms], 46		TC [ms], 48	
		TE [ms], 44	
	- p -	TH [man], 22	
P [man], 23		TL [ms], 34	
P1 [ms], 34		TP [man], 23	
PD [man], 25		TS [ms], 44	
PE [ms], 45			- u -
PN [man], 28		UC [man], 26	
Pn [man], 28		UL [ms], 39	
PP			- v -
[man], 23			
[ms], 35			
PS [ms], 45		VE [man], 28	
PT		VS [man], 28	
[man], 27			- x -
[ms], 46			
PX [ms], 48			
	- q -	XA [ms], 47	
QE [ms], 36		XE [ms], 47	
QP [ms], 36		XP [ms], 36	
QS [ms], 36		XS [ms], 47	
	- r -		
R			
[man], 28			
[ms], 39			
RB [man], 25			
RD [ms], 43			
RE			
[man], 24			
[ms], 42			
RI [man], 25			
RN [man], 28			

## G String Index

---

[ms]		MONTH6 [ms], <a href="#">49</a>
, <a href="#">50</a>		MONTH7 [ms], <a href="#">49</a>
, <a href="#">50</a>		MONTH8 [ms], <a href="#">49</a>
! [ms], <a href="#">50</a>		MONTH9 [ms], <a href="#">49</a>
' [ms], <a href="#">50</a>		
* [ms], <a href="#">45</a>		
- [ms], <a href="#">49</a>		- o -
. [ms], <a href="#">50</a>		o [ms], <a href="#">50</a>
3 [ms], <a href="#">50</a>		
8 [ms], <a href="#">50</a>		- q -
? [ms], <a href="#">50</a>		Q [ms], <a href="#">49</a>
^ [ms], <a href="#">50</a>		q [ms], <a href="#">51</a>
_ [ms], <a href="#">50</a>		
‘ [ms], <a href="#">50</a>		- r -
{ [ms], <a href="#">39</a>		R [man], <a href="#">26</a>
} [ms], <a href="#">39</a>		REFERENCES [ms], <a href="#">49</a>
	- a -	RF [ms], <a href="#">46</a>
ABSTRACT [ms], <a href="#">49</a>		RH [ms], <a href="#">46</a>
Ae [ms], <a href="#">51</a>		rq [man], <a href="#">26</a>
ae [ms], <a href="#">51</a>		
	- c -	- s -
		S [man], <a href="#">26</a>
CF [ms], <a href="#">46</a>		SN [ms], <a href="#">38</a>
CH [ms], <a href="#">46</a>		SN-DOT [ms], <a href="#">38</a>
	- d -	SN-NO-DOT [ms], <a href="#">38</a>
		SN-STYLE [ms], <a href="#">38</a>
d- [ms], <a href="#">51</a>		
D- [ms], <a href="#">50</a>		- t -
	- h -	th [ms], <a href="#">50</a>
		Th [ms], <a href="#">50</a>
HF [man], <a href="#">26</a>		Tm [man], <a href="#">26</a>
	- l -	TOC [ms], <a href="#">49</a>
LF [ms], <a href="#">46</a>		- u -
LH [ms], <a href="#">46</a>		U [ms], <a href="#">49</a>
lq [man], <a href="#">26</a>		
	- v -	
		v [ms], <a href="#">50</a>
	- m -	
		- w -
MONTH1 [ms], <a href="#">49</a>		www-image-template [grohtml], <a href="#">293</a>
MONTH10 [ms], <a href="#">49</a>		
MONTH11 [ms], <a href="#">49</a>		
MONTH12 [ms], <a href="#">49</a>		
MONTH2 [ms], <a href="#">49</a>		
MONTH3 [ms], <a href="#">49</a>		
MONTH4 [ms], <a href="#">49</a>		
MONTH5 [ms], <a href="#">49</a>		

## J Program and File Index

---

- a -
- an.tmac, [21](#)
- c -
- changebar, [216](#)
- composite.tmac, [159](#)
- cp1047.tmac, [107](#)
- d -
- DESC, [152](#), [155](#), [156](#), [157](#), [160](#), [164](#)
- DESC
  - and use\_charnames\_in\_special, [214](#)
  - and font mounting, [156](#)
  - file format, [307](#)
- ditroff, [3](#)
- e -
- ec.tmac, [107](#)
- eqn, [44](#)
- f -
- freeeuro.pfa, [107](#)
- g -
- gchem, [7](#)
- geqn, [7](#)
- geqn, invocation in manual pages, [26](#)
- ggrn, [7](#)
- gpic, [7](#)
- grap, [7](#)
- grefer, [7](#)
- grefer, invocation in manual pages, [26](#)
- groff, [7](#)
- grog, [15](#)
- grohtml, [25](#)
- gsoelim, [7](#)
- gtbl, [7](#)
- gtbl, invocation in manual pages, [26](#)
- gtroff, [7](#)
- h -
- hyphen.us, [132](#)
- l -
- latin1.tmac, [107](#)
- latin2.tmac, [107](#)
- latin5.tmac, [107](#)
- latin9.tmac, [107](#)
- less, [287](#)
- m -
- makeindex, [20](#)
- man, invocation of preprocessors, [26](#)
- man-old.tmac, [21](#)
- man.local, [22](#), [26](#)
- man.tmac, [21](#)
- man.ultrix, [27](#)
- n -
- nrchbar, [216](#)
- p -
- papersize.tmac, [14](#)
- perl, [212](#)
- pic, [44](#)
- post-grohtml, [11](#)
- pre-grohtml, [11](#)
- preconv, [7](#)
- r -
- refer, [44](#)
- s -
- soelim, [219](#)
- t -
- tbl, [44](#)
- trace.tmac, [184](#), [184](#)
- troffrc, [10](#), [14](#), [132](#), [133](#), [144](#), [145](#)
- troffrc-end, [10](#), [132](#), [133](#), [144](#)
- tty.tmac, [144](#)

## K Concept Index

- 
- \!
  - and copy-in mode, 204
  - in top-level diversion, 204
  - used as delimiter, 116, 116
  - \0, used as delimiter, 116
  - 8-bit input, 310
  - \!
  - and output request, 205
  - and `trnt`, 143
  - incompatibilities with AT&T `troff`, 224, 225
  - \$ -
  - \\$, when reading text for a macro, 184
  - % -
  - \%
  - and translations, 142
  - as delimiter, 116
  - used as delimiter, 116, 116
  - \%
  - following `\X` or `\Y`, 131
  - in `\X`, 214
  - incompatibilities with AT&T `troff`, 224
  - & -
  - \&
  - and glyph definitions, 162
  - and translations, 142
  - as delimiter, 116
  - at end of sentence, 106
  - escaping control characters, 113
  - used as delimiter, 116
  - \&
  - in `\X`, 214
  - incompatibilities with AT&T `troff`, 224
  - ' -
  - \'
  - and translations, 142
  - as a comment, 117
  - at end of sentence, 106, 161
  - delimiting arguments, 115
  - used as delimiter, 116, 116
  - \', incompatibilities with AT&T `troff`, 224
  - ( -
  - \(
  - and translations, 142
  - as delimiter, 116
  - starting a two-character identifier, 111, 115
  - ) -
  - )
  - as delimiter, 116
  - at end of sentence, 106, 161
  - used as delimiter, 116
  - \), in `\X`, 214
  - \* -
  - \\*
  - and warnings, 222
  - as delimiter, 116
  - at end of sentence, 106, 161
  - when reading text for a macro, 184
  - \\*, incompatibilities with AT&T `troff`, 223
  - + -
  - +
  - and page motion, 110
  - as delimiter, 116
  - - -
  - 
  - and page motion, 110
  - and translations, 142
  - as delimiter, 116
  - used as delimiter, 116, 116
  - \-, incompatibilities with AT&T `troff`, 224
  - . -
  - ., as delimiter, 116
  - .h register, difference to `n1`, 203
  - .ps register, in comparison with `.psr`, 172
  - .S register, Plan 9 alias for `.tabs`, 138
  - .s register, in comparison with `.sr`, 172
  - .t register, and diversions, 198
  - .tabs register, Plan 9 alias (`.s`), 138
  - .V register, and `vs`, 171
  - :-
  - :
  - as delimiter, 116
  - used as delimiter, 116, 116
  - \:, in `\X`, 214
  - < -
  - <, as delimiter, 116

- <colon>, as delimiter, 116
- = -
- =, as delimiter, 116
- > -
- >, as delimiter, 116
- ? -
- \?
- and copy-in mode, 178, 204
- in top-level diversion, 205
- used as delimiter, 116
- \?, incompatibilities with AT&T `troff`, 225
- @ -
- \D'f ...' and horizontal resolution, 193
- \SP, used as delimiter, 116
- \SP
- difference to \ , 114
- incompatibilities with AT&T `troff`, 224
- \<colon>, used as delimiter, 116, 116
- \<colon>, in \X, 214
- \{, used as delimiter, 116, 116
- \{, incompatibilities with AT&T `troff`, 224
- \}
- and warnings, 222
- used as delimiter, 116, 116
- \}, incompatibilities with AT&T `troff`, 224
- [ -
- \[
- and translations, 142
- starting an identifier, 111, 115
- [, macro names starting with, and `refer`, 111
- ] -
- ]
- as part of an identifier, 111
- at end of sentence, 106, 161
- ending an identifier, 111, 115
- ], macro names starting with, and `refer`, 111
- ^ -
- \^, used as delimiter, 116
- \^, incompatibilities with AT&T `troff`, 224
- \_ -
- \\_
- and translations, 142
- used as delimiter, 116, 116
- \\_, incompatibilities with AT&T `troff`, 224
- ' -
- \`
- and translations, 142
- used as delimiter, 116, 116
- \', incompatibilities with AT&T `troff`, 224
- a -
- \A, allowed delimiters, 116
- \a
- and copy-in mode, 138
- and translations, 142
- used as delimiter, 116
- aborting (`ab`), 219
- absolute position operator (`|`), 110
- accent marks [`ms`], 49
- access of postprocessor, 214
- accessing unnamed glyphs with \N, 310
- activating
- kerning (`kern`), 167
- ligatures (`lg`), 167
- track kerning (`tkf`), 167
- ad request
- and hyphenation margin, 132
- and hyphenation space, 133
- adjusting, 105
- and filling, manipulating, 124
- adjustment mode register (`.j`), 126
- adobe glyph list (AGL), 158
- AGL (adobe glyph list), 158
- alias
- diversion, creating (`als`), 177
- diversion, removing (`rm`), 177
- macro, creating (`als`), 177
- macro, removing (`rm`), 177
- number register, creating (`aln`), 120
- string, creating (`als`), 177
- string, removing (`rm`), 177
- `als` request, and \S0, 186
- `am`, `aml`, `ami` requests, and warnings, 222
- \ , and translations, 142
- annotations, 19
- appending to
- a diversion (`da`), 202
- a file (`opena`), 213
- a macro (`am`), 184
- a string (`as`), 176
- arc, drawing (\D'a ...'), 193
- argument delimiting characters, 115
- arguments
- and compatibility mode, 218
- of strings, 173
- to macros, and tabs, 114
- to requests and macros, 114

arguments, macro ( $\backslash\$\$ ), 185  
 arithmetic operators, 109  
 artificial fonts, 164  
 /, as delimiter, 116  
 ASCII, output encoding, 10  
*asciify* request, and *writem*, 213  
 ASCII approximation output register ( $\backslash.A$ ), 8, 124  
 assigning formats (*af*), 121  
 assignments  
   indirect, 120  
   nested, 120  
*as*, *as1*  
   requests, and comments, 116  
   requests, and warnings, 222  
   , at end of sentence, 106, 161  
 AT&T *troff*, *ms* macro package differences, 51  
 auto-increment, 120  
 auto-increment, and *ig* request, 118  
 available glyphs, list (*groff\_char(7)* man page), 158  
 $\backslash A$ , incompatibilities with AT&T *troff*, 224

## - b -

$\backslash B$ , allowed delimiters, 116  
 $\backslash b$   
   limitations, 195  
   possible quote characters, 115  
 background color name register ( $\backslash.M$ ), 210  
 backslash, printing ( $\backslash$ ,  $\backslash e$ ,  $\backslash E$ ,  $\backslash[rs]$ ), 116, 225  
 backspace character, 110  
   and translations, 142  
 backtrace of input stack (*backtrace*), 220  
 baseline, 169  
 basic unit (*u*), 108  
 basics of macros, 16  
*bd* request  
   and font styles, 154  
   and font translations, 153  
*bd* request, incompatibilities with AT&T *troff*, 224  
 begin of conditional block ( $\backslash\{$ ), 180  
 beginning diversion (*di*), 202  
 blank  
   line, 106, 113  
   line traps, 199  
   lines, disabling, 135  
 blank line  
   (*sp*), 17  
   macro (*blm*), 106, 113, 199  
 block, conditional  
   begin ( $\backslash\{$ ), 180  
   end ( $\backslash\}$ ), 180  
 bold face  
   [*man*], 24  
   imitating (*bd*), 166  
 bottom margin, 149

bounding box, 217  
 box rule glyph ( $\backslash[br]$ ), 191  
*boxa* request, and *dn* (*dl*), 203  
 box, *boxa* requests, and warnings, 222  
*bp* request  
   and top-level diversion, 150  
   causing implicit linebreak, 124  
*bp* request  
   and traps (*.pe*), 198  
   using + and -, 110  
 break, 16, 124  
   implicit, 106  
 break (*br*), 18  
 break request, in a *while* loop, 181  
*br* glyph, and *cflags*, 161  
 built-in registers, 122  
 bulleted list, example markup [*ms*], 40

## - c -

$\backslash c$   
   and fill mode, 148  
   and no-fill mode, 148  
   unit, 108  
   used as delimiter, 116, 116  
 $\backslash C$   
   allowed delimiters, 116  
   and translations, 142  
 calling convention of preprocessors, 26  
 capabilities of *groff*, 4  
*ce* request, causing implicit linebreak, 124  
 centered text, 125  
 centering lines (*ce*), 17, 127  
 centimeter unit (*c*), 108  
*ce* request, difference to '*.ad c*', 125  
*cf* request  
   and copy-in mode, 210  
   causing implicit linebreak, 124  
 changing format, and read-only registers, 122  
 changing  
   font family (*fam*,  $\backslash F$ ), 154  
   font position ( $\backslash f$ ), 156  
   font style (*sty*), 154  
   fonts (*ft*,  $\backslash f$ ), 152  
   the font height ( $\backslash H$ ), 164  
   the font slant ( $\backslash S$ ), 165  
   the page number character (*pc*), 150  
   trap location (*ch*), 197  
   type sizes (*ps*,  $\backslash s$ ), 169  
   vertical line spacing (*vs*), 171  
*char* request  
   and soft hyphen character, 133  
   and translations, 142  
 character, 156  
   backspace, 110  
   backspace, and translations, 142  
   classes, 163  
   escape, while defining glyph, 162  
   leader, and translations, 142

- newline, 116
- newline, and translations, 142
- space, 116
- special, 142
- tab, 116
- tab, and translations, 142
- tabulator, 106
- translations, 140
- transparent, 106, 161
- whitespace, 110
- character
  - class (`class`), 163
  - control (`.`), 112
  - control, changing (`cc`), 140
  - defining (`char`), 162
  - defining fallback (`fchar`, `fschar`, `schar`), 162
  - escape, changing (`ec`), 140
  - field delimiting (`fc`), 139
  - field padding (`fc`), 139
  - hyphenation (`\%`), 130
  - leader repetition (`lc`), 139
  - leader, non-interpreted (`\a`), 138
  - named (`\C`), 159
  - no-break control (`'`), 112
  - no-break control, changing (`c2`), 140
  - properties (`cflags`), 160
  - soft hyphen, setting (`shc`), 133
  - tab repetition (`tc`), 138
  - tab, non-interpreted (`\t`), 136
  - zero width space (`\&`), 113, 167, 191
- characters
  - argument delimiting, 115
  - end-of-sentence, 160
  - hyphenation, 161
  - invalid input, 110
  - overlapping, 161
  - special, 287
- characters
  - input, and output glyphs, compatibility with AT&T `troff`, 224
  - invalid for `trf` request, 211
  - unnamed, accessing with `\N`, 310
- `char` request, used with `\N`, 160
- `chem`, the program, 263
- circle
  - drawing (`\D'c ...'`), 192
  - solid, drawing (`\D'C ...'`), 192
- class of characters (`class`), 163
- classes, character, 163
- closing file (`close`), 213
- code, hyphenation (`hcode`), 132
- color, default, 208
- color name
  - background, register (`.M`), 210
  - drawing, register (`.m`), 209
  - fill, register (`.M`), 210
- colors, 208
- colors, fill, unnamed (`\D'F...'`), 194
- command prefix, 12
- command-line options, 8
- commands, embedded, 112
- comments, 116
  - in font files, 310
  - lining up with tabs, 117
- comments, with `ds`, 174
- common
  - features, 18
  - name space of macros, diversions, and strings, 174
- comparison
  - of strings, 178
  - operators, 109
- compatibility mode, 223, 223
  - and parameters, 218
- composite glyph names, 158
- conditional output for terminal (TTY), 178
- conditional
  - block, begin (`\{`), 180
  - block, end (`\}`), 180
  - page break (`ne`), 150
- conditionals and loops, 178
- consecutive hyphenated lines (`hlm`), 130
- constant glyph space mode (`cs`), 166
- contents, table of, 20, 139
- continuation
  - input line (`\`), 148
  - output line (`\c`), 148
- `continue` request, in a `while` loop, 181
- continuous underlining (`cu`), 165
- control
  - line, 147
  - page, 150
  - sequences, for terminals, 287
- control character
  - (`.`), 112
  - changing (`cc`), 140
  - no-break (`'`), 112
  - no-break, changing (`c2`), 140
- conventions for input, 106
- copy mode, 184
- copy-in mode, 184
  - and macro arguments, 185
- copy-in mode
  - and `\!`, 204
  - and `\?`, 178, 204
  - and `\a`, 138
  - and `cf` request, 210
  - and device request, 214
  - and `\E`, 141
  - and `ig` request, 118
  - and length request, 176
  - and output request, 205
  - and `\t`, 136
  - and `tm` request, 219
  - and `tml` request, 219

- and `tmc` request, 219
- and `trf` request, 210
- and `\V`, 214
- and `write` request, 213
- and `writec` request, 213
- and `writem` request, 213
- copying environment (`evc`), 207
- correction
  - between italic and roman glyph (`\/, \r`), 167
  - italic (`\/`), 167
  - left italic (`\r`), 168
- cover page macros, `[ms]`, 34
- `cp` request, and glyph definitions, 162
- `cp1047`
  - input encoding, 107
  - output encoding, 11
- `cq` glyph, at end of sentence, 106, 161
- creating
  - alias, for diversion (`als`), 177
  - alias, for macro (`als`), 177
  - alias, for number register (`aln`), 120
  - alias, for string (`als`), 177
  - new characters (`char`), 162
- credits, 6
- `cs` request
  - and font styles, 154
  - and font translations, 153
  - with fractional type sizes, 172
- `cs` request, incompatibilities with AT&T `troff`, 224
- current
  - directory, 13
  - time, 212
- current
  - input file name register (`.F`), 122
  - page number (`%`), 150
  - time, hours (`hours`), 123
  - time, minutes (`minutes`), 123
  - time, seconds (`seconds`), 123
  - vertical position (`n1`), 151
- `\c`, incompatibilities with AT&T `troff`, 224

- d -

- `\d`, used as delimiter, 116
- `\D`, allowed delimiters, 116
- `da` request, and warnings, 222, 222
- date
  - day of the month register (`dy`), 123
  - day of the week register (`dw`), 123
  - month of the year register (`mo`), 123
  - year register (`year, yr`), 123
- day of
  - the month register (`dy`), 123
  - the week register (`dw`), 123
- `da` request, and `dn` (`d1`), 203
- debugging, 219

- default
  - color, 208
  - units, 108
- default indentation
  - `[man]`, 25
  - resetting `[man]`, 24
- defining
  - character (`char`), 162
  - character class (`class`), 163
  - fallback character (`fchar, fschar, schar`), 162
  - glyph (`char`), 162
  - symbol (`char`), 162
- delayed text, 19
- delimited arguments, incompatibilities with AT&T `troff`, 224
- delimiting characters for arguments, 115
- delimiting character, for fields (`fc`), 139
- depth, of last glyph (`.cdp`), 207
- DESC file, format, 307
- device resolution, 308
- device request, and copy-in mode, 214
- devices for output, 5, 287
- `de`
  - `del, dei` requests, and warnings, 222
  - request, and `while`, 181
- `dg` glyph, at end of sentence, 106, 161
- `di` request, and warnings, 222, 222
- differences in implementation, 223
- digit width space (`\0`), 189
- digits, and delimiters, 116
- dimensions, line, 144
- directories for
  - fonts, 13
  - macros, 12
- directory
  - current, 13
  - for `tmac` files, 13
  - home, 13
  - platform-specific, 13
  - site-specific, 13, 13
- disabling `\(eo)`, 140
  - hyphenation (`\%`), 130
- discardable horizontal space, 127
- discarded space in traps, 134
- displays, 19
- displays
  - and footnotes `[ms]`, 46
  - `[ms]`, 43
- distance to next trap register (`.t`), 197
- `ditroff`, the program, 3
- diversion
  - nested, 202
  - stripping final newline, 175
  - top-level, 201
  - traps, 198
- diversion
  - appending (`da`), 202
  - beginning (`di`), 202

- creating alias (`als`), 177
  - ending (`di`), 202
  - name register (`.z`), 202
  - removing (`rm`), 177
  - removing alias (`rm`), 177
  - renaming (`rn`), 177
  - top-level, and `\!`, 204
  - top-level, and `\?`, 205
  - top-level, and `bp`, 150
  - trap, setting (`dt`), 198
  - unformatting (`asciify`), 205
  - vertical position in, register (`.d`), 202
- diversions, 201
  - and traps, 198
  - shared name space with macros and strings, 174
- `dl` register, and `da` (`boxa`), 203
- `dn` register, and `da` (`boxa`), 203
- documents
  - multi-file, 219
  - structuring the source code, 112
- double quote, in a macro argument, 114
- double-spacing
  - (`ls`), 17, 134
  - (`vs`, `pvs`), 171
- drawing requests, 191
- drawing
  - a circle (`\D'c ...'`), 192
  - a line (`\D'l ...'`), 192
  - a polygon (`\D'p ...'`), 193
  - a solid circle (`\D'C ...'`), 192
  - a solid ellipse (`\D'E ...'`), 193
  - a solid polygon (`\D'P ...'`), 193
  - a spline (`\D' ...'`), 193
  - an arc (`\D'a ...'`), 193
  - an ellipse (`\D'e ...'`), 193
  - color name register (`.m`), 209
  - horizontal lines (`\l`), 191
  - vertical lines (`\L`), 191
- `ds` request
  - and comments, 174
  - and double quotes, 115
  - and leading spaces, 174
- `ds`, `ds1`
  - requests, and comments, 116
  - requests, and warnings, 222
- dumping
  - environments (`pev`), 220
  - number registers (`pnr`), 220
  - symbol table (`pm`), 220
  - traps (`ptr`), 220
- e -
- `\e`
  - and glyph definitions, 162
  - and translations, 142
  - used as delimiter, 116, 116
- `\E`
  - and copy-in mode, 141
  - used as delimiter, 116
- EBCDIC
  - encoding, 106
  - encoding of a tab, 136
  - encoding of backspace, 110
  - input encoding, 107
  - output encoding, 11
- `e1` request, and warnings, 222
- ellipse
  - drawing (`\D'e ...'`), 193
  - solid, drawing (`\D'E ...'`), 193
- em unit (`m`), 108
- embedded commands, 112
- embedding
  - PDF, 289
  - PostScript, 288
- embolding of special fonts, 166
- empty line, 106
- empty
  - line (`sp`), 17
  - space before a paragraph [`man`], 25
- em glyph, and `cflags`, 161
- en unit (`n`), 108
- enabling vertical position traps (`vpt`), 195
- encoding
  - EBCDIC, 106
  - input, `cp1047`, 107
  - input, EBCDIC, 107
  - input, latin-1 (ISO 8859-1), 107
  - input, latin-2 (ISO 8859-2), 107
  - input, latin-5 (ISO 8859-9), 107
  - input, latin-9 (latin-0, ISO 8859-15), 107
  - output, ASCII, 10
  - output, `cp1047`, 11
  - output, EBCDIC, 11
  - output, latin-1 (ISO 8859-1), 10
  - output, utf-8, 10
- end of conditional block (`\}`), 180
- end-of-input traps, 200
- end-of-input
  - macro (`em`), 200
  - trap, setting (`em`), 200
- end-of-sentence characters, 160
- ending diversion (`di`), 202
- environment variables, 11
- environment
  - copying (`evc`), 207
  - dimensions of last glyph (`.w`, `.cht`, `.cdp`, `.csk`), 207
  - number/name register (`.ev`), 206
  - previous line length (`.n`), 207
  - switching (`ev`), 206
- environments, 205
- environments, dumping (`pev`), 220
- `eqn`, the program, 226

- equations [ms], 44
  - escape character, while defining glyph, 162
  - escape character, changing (ec), 140
  - escapes, 115
  - escaping newline characters, in strings, 174
  - ex request, use in debugging, 219
  - example markup, title page, 35
  - example markup
    - bulleted list [ms], 40
    - glossary-style list [ms], 40
    - multi-page table [ms], 45
    - numbered list [ms], 40
  - examples of invocation, 14
  - exiting (ex), 219
  - expansion of strings (\\*), 173
  - explicit hyphen (\%), 130
  - expression
    - limitation of logical not in, 109
    - order of evaluation, 110
  - expressions, 109
    - and space characters, 110
  - extra spaces, 105
  - extra
    - post-vertical line space (\x), 171
    - post-vertical line space register (.a), 135
    - pre-vertical line space (\x), 171
  - extremum operators (>?, <?), 110
  - ex request, used with nx and rd, 211
  - \e, incompatibilities with AT&T troff, 225
- f -
- \F, and
    - changing fonts, 152
    - font positions, 156
  - \f
    - and font translations, 153
    - unit, 108
    - unit, and colors, 209
  - factor, zoom, of a font (fzoom), 153
  - fallback
    - character, defining (fchar, fschar, schar), 162
    - glyph, removing definition (rchar, rfschar), 163
  - fam request
    - and changing fonts, 152
    - and font positions, 156
  - families, font, 153
  - features, common, 18
  - fi request, causing implicit linebreak, 124
  - field
    - delimiting character (fc), 139
    - padding character (fc), 139
  - fields, 139
    - and tabs, 136
  - figures [ms], 44
  - file formats, 294
  - file
    - appending to (opena), 213
    - closing (close), 213
    - inclusion (so), 210
    - opening (open), 213
    - processing next (nx), 211
    - writing to (write, writec), 213
  - files
    - font, 306
    - macro, searching, 13
  - fill mode, 106, 126, 222
  - fill
    - color name register (.M), 210
    - colors, unnamed (\D'F...'), 194
    - mode (fi), 124
    - mode, and \c, 148
  - filling, 105
    - and adjusting, manipulating, 124
  - final newline, stripping in diversions, 175
  - fl request, causing implicit linebreak, 124
  - floating keep, 19
  - flush output (fl), 220
  - Font File
    - #, 310
    - , 310
    - biggestfont, 309
    - charset, 307, 310
    - family, 152, 156, 307
    - fonts, 157, 164, 307
    - hor, 307
    - image\_generator, 307
    - kernpairs, 311
    - ligatures, 309
    - name, 309
    - paperlength, 307
    - papersize, 307
    - paperwidth, 307
    - pass\_filenames, 308
    - postpro, 308
    - prepro, 308
    - print, 308
    - res, 308
    - sizes, 308
    - sizescale, 308
    - slant, 309
    - spacewidth, 309
    - spare1, 309
    - spare2, 309
    - special, 166, 309
    - styles, 152, 155, 156, 308
    - tcommand, 308
    - unicode, 308
    - unitwidth, 309
    - unscaled\_charwidths, 309
    - use\_charnames\_in\_special, 214, 309
    - vert, 309

- font
    - description file, format, 307, 309
    - directories, 13
    - families, 153
    - file, format, 309
    - files, 306
    - files, comments, 310
    - optical size, 153
    - path, 13
    - positions, 155
    - styles, 153
  - font
    - family, changing (`fam`, `\F`), 154
    - for underlining (`uf`), 165
    - height, changing (`\H`), 164
    - magnification (`fzoom`), 153
    - mounting (`fp`), 155
    - position register (`.f`), 155
    - position, changing (`\f`), 156
    - previous (`ft`, `\f[]`, `\fP`), 152
    - selection [`man`], 24
    - slant, changing (`\S`), 165
    - style, changing (`sty`), 154
    - translation (`ftx`), 153
    - zoom factor (`fzoom`), 153
  - fonts, 152, 152
    - artificial, 164
    - PostScript, 153
    - searching, 13
    - special, 164
  - fonts, changing (`ft`, `\f`), 152
  - footers, 149, 196
  - footers [`ms`], 46
  - footnotes, 19
  - footnotes
    - and displays [`ms`], 46
    - and keeps [`ms`], 46
    - [`ms`], 45
  - form letters, 211
  - format of
    - font description file, 307
    - font description files, 309
    - font files, 309
  - format of register (`\g`), 122
  - formats, file, 294
  - formats, assigning (`af`), 121
  - `fp` request, and font translations, 153
  - `fp` request, incompatibilities with AT&T `troff`, 224
  - fractional
    - point sizes, 172, 224
    - type sizes, 172, 224
  - french-spacing, 105
  - `fspecial` request
    - and font styles, 154
    - and font translations, 153
    - and glyph search order, 157
    - and imitating bold, 166
  - `ft` request, and font translations, 153
  - `\f`, incompatibilities with AT&T `troff`, 224
- g -
- `gchem`, the program, 263
  - `geqn`, the program, 226
  - GGL (groff glyph list), 158, 163
  - `ggrn`, the program, 257
  - glossary-style list, example markup [`ms`], 40
  - glyph, 156
    - constant space, 166
    - for line drawing, 191
    - for line drawing, 191
    - names, composite, 158
  - glyph
    - box rule (`\[br]`), 191
    - defining (`char`), 162
    - for margins (`mc`), 216
    - italic correction (`\/`), 167
    - last, dimensions (`.w`, `.cht`, `.cdp`, `.csk`), 207
    - leader repetition (`lc`), 139
    - left italic correction (`\,`), 168
    - numbered (`\N`), 142, 160
    - pile (`\b`), 195
    - properties (`cflags`), 160
    - removing definition (`rchar`, `rfschar`), 163
    - soft hyphen (`hy`), 133
    - tab repetition (`tc`), 138
    - underscore (`\[ru]`), 191
  - glyphs
    - available, list (`groff_char(7)` man page), 158
    - unnamed, 160
  - glyphs
    - output, and input characters, compatibility with AT&T `troff`, 224
    - overstriking (`\o`), 190
    - unnamed, accessing with `\N`, 310
  - GNU-specific register (`.g`), 124
  - `gpic`, the program, 248
  - `grap`, the program, 263
  - gray shading (`\D'f ...'`), 193
  - `grefer`, the program, 270
  - `grn`, the program, 257
  - `grodvi`
    - invoking, 290
    - the program, 290
  - groff glyph list (GGL), 158, 163
  - groff
    - what is it?, 2
    - capabilities, 4
    - invocation, 7
  - groff, and `pi` request, 212
  - `grohtml`
    - invoking, 292
    - registers and strings, 293
    - the program, 11, 291

gro1bp  
   invoking, 291  
   the program, 291  
 gro1j4  
   invoking, 290  
   the program, 290  
 gropdf  
   invoking, 289  
   the program, 289  
 grops  
   invoking, 288  
   the program, 288  
 grotty  
   invoking, 287  
   the program, 287  
 gsoelim, the program, 282  
 gtbl, the program, 237  
 gtroff  
   interactive use, 220  
   output, 294  
   reference, 105  
 gtroff  
   identification register (.g), 124  
   process ID register (\$\$), 124  
 gxdtview  
   invoking, 293  
   the program, 293

- h -

\H  
   allowed delimiters, 116  
   with fractional type sizes, 172  
 \h, allowed delimiters, 116  
 hanging indentation [man], 24  
 hcode request, and glyph definitions, 162  
 headers, 149, 196  
 headers [ms], 46  
 height  
   font, changing (\H), 164  
   of last glyph (.cht), 207  
 high-water mark register (.h), 203  
 history, 2  
 home directory, 13  
 horizontal  
   discardable space, 127  
   resolution, 307  
   space, unformatting, 175  
 horizontal  
   input line position register (hp), 190  
   input line position, saving (\k), 190  
   line, drawing (\l), 191  
   motion (\h), 189  
   output line position register (.k), 190  
   resolution register (.H), 122  
   space (\h), 189  
 hours, current time (hours), 123

hpf request, and hyphenation language, 133  
 hw request, and hyphenation language, 133  
 hw request, and hy restrictions, 130  
 hyphen, explicit (\%), 130  
 hyphenated lines, consecutive (hlm), 130  
 hyphenating characters, 161  
 hyphenation, 105  
   manipulating, 128  
 hyphenation  
   character (\%), 130  
   code (hcode), 132  
   disabling (\%), 130  
   language register (.hla), 133  
   margin (hym), 132  
   margin register (.hym), 133  
   patterns (hpf), 131  
   restrictions register (.hy), 129  
   space (hys), 133  
   space register (.hys), 133  
 hy glyph, and cflags, 161  
 \H  
   incompatibilities with AT&T troff, 224  
   using + and -, 110

- i -

i unit, 108  
 IBM cp1047  
   input encoding, 107  
   output encoding, 11  
 identifiers, 110  
   undefined, 111  
 ie request  
   and font translations, 153  
   and warnings, 222  
   operators to use with, 178  
 if request  
   and font translations, 153  
   operators to use with, 178  
 if-else, 180  
 if request, and the '!' operator, 109  
 ig request  
   and auto-increment, 118  
   and copy-in mode, 118  
 imitating bold face (bd), 166  
 implementation differences, 223  
 implicit  
   breaks of lines, 106  
   line breaks, 106  
   , in a macro argument, 114  
   request, causing implicit linebreak, 124  
 inch unit (i), 108  
 including a file (so), 210  
 incompatibilities with AT&T troff, 223  
 increment  
   automatic, 120  
   value without changing the register, 121

- indentation
    - (in), 145
    - resetting to default [man], 24
  - index, in macro package, 20
  - indicator, scaling, 108
  - indirect assignments, 120
  - input
    - 8-bit, 310
    - and output requests, 210
    - characters, invalid, 110
    - conventions, 106
    - encoding, cp1047, 107
    - encoding, EBCDIC, 107
    - encoding, latin-1 (ISO 8859-1), 107
    - encoding, latin-2 (ISO 8859-2), 107
    - encoding, latin-5 (ISO 8859-9), 107
    - encoding, latin-9 (latin-9, ISO 8859-15), 107
    - level in delimited arguments, 224
    - line traps, 198
    - stack, setting limit, 221
    - token, 217
  - input
    - characters and output glyphs, compatibility with AT&T `troff`, 224
    - file name, current, register (`.F`), 122
    - line continuation (`\`), 148
    - line number register (`.c, c.`), 123
    - line number, setting (`lf`), 219
    - line position, horizontal, saving (`\k`), 190
    - line trap, setting (`it`), 198
    - line traps and interrupted lines (`itc`), 199
    - line, horizontal position, register (`hp`), 190
    - stack, backtrace (`backtrace`), 220
    - standard, reading from (`rd`), 211
  - inserting horizontal space (`\h`), 189
  - installation, 312
  - interactive use of `gtroff`, 220
  - intermediate output, 294
  - interpolating registers (`\n`), 120
  - interpolation of strings (`\*`), 173
  - interrupted line, 148
  - interrupted
    - line register (`.int`), 148
    - lines and input line traps (`itc`), 199
  - introduction, 2
  - invalid input characters, 110
  - invalid characters for `trf` request, 211
  - invocation examples, 14
  - invoking
    - `grodvi`, 290
    - `groff`, 7
    - `grohtml`, 292
    - `grolbp`, 291
    - `grolj4`, 290
    - `gropdf`, 289
    - `grops`, 288
    - `grotty`, 287
    - `gxditview`, 293
  - `in` request, using + and -, 110
  - i/o, 210
  - ISO
    - 6249 SGR, 287
    - 8859-1 (latin-1), input encoding, 107
    - 8859-1 (latin-1), output encoding, 10
    - 8859-15 (latin-9, latin-0), input encoding, 107
    - 8859-2 (latin-2), input encoding, 107
    - 8859-9 (latin-5), input encoding, 107
  - italic
    - correction (`\/`), 167
    - fonts [man], 25
    - glyph, correction after roman glyph (`\,`), 168
    - glyph, correction before roman glyph (`\/`), 167
- j -
- justifying text, 124
  - justifying text (`rtj`), 128
- k -
- keep, 19
    - floating, 19
  - keeps
    - and footnotes [ms], 46
    - [ms], 43
  - kerning
    - and ligatures, 166
    - track, 167
  - kerning
    - activating (`kern`), 167
    - enabled register (`.kern`), 167
- l -
- `\l`
    - allowed delimiters, 116
    - and glyph definitions, 162
  - `\L`
    - allowed delimiters, 116
    - and glyph definitions, 162
  - landscape page orientation, 14
  - last glyph, dimensions (`.w, .cht, .cdp, .csk`), 207
  - last-requested point size registers (`.psr, .sr`), 172
  - latin-1 (ISO 8859-1), input encoding, 107
  - latin-1 (ISO 8859-1), output encoding, 10
  - latin-2 (ISO 8859-2), input encoding, 107
  - latin-5 (ISO 8859-9), input encoding, 107
  - latin-9 (latin-0, ISO 8859-15), input encoding, 107
  - layout
    - line, 144
    - page, 148

- lc request, and glyph definitions, 162
  - leader character, 138
    - and translations, 142
  - leader
    - character, non-interpreted (`\a`), 138
    - repetition character (`lc`), 139
  - leaders, 138
  - leading, 169
    - spaces, 105
    - spaces traps, 199
  - leading spaces
    - macro (`lsm`), 106, 199
    - with `ds`, 174
  - left
    - italic correction (`\,`), 168
    - margin (`po`), 145
    - margin, how to move [`man`], 24
  - length
    - of a string (`length`), 176
    - of line (`ll`), 145
    - of page (`pl`), 149
    - of previous line (`.n`), 207
    - of title line (`lt`), 149
    - request, and copy-in mode, 176
  - letters, form, 211
  - level of warnings (`warn`), 221
  - ligature, 156
  - ligatures and kerning, 166
  - ligatures
    - activating (`lg`), 167
    - enabled register (`.lg`), 167
  - limitations of `\b` escape, 195
  - line
    - blank, 106
    - break, 16, 106, 124
    - control, 147
    - dimensions, 144
    - drawing glyph, 191, 191
    - implicit breaks, 106
    - interrupted, 148
    - layout, 144
  - line
    - break (`br`), 18
    - breaks, with vertical space [`man`], 24
    - breaks, without vertical space [`man`], 24
    - drawing (`\D'l ...'`), 192
    - empty (`sp`), 17
    - horizontal, drawing (`\l`), 191
    - indentation (`in`), 145
    - input, continuation (`\`), 148
    - input, horizontal position, register (`hp`), 190
    - input, horizontal position, saving (`\k`), 190
    - length (`ll`), 145
    - length register (`.l`), 147
    - length, previous (`.n`), 207
    - number, input, register (`.c, c.`), 123
    - number, output, register (`ln`), 123
    - numbers, printing (`nm`), 214
    - output, continuation (`\c`), 148
    - output, horizontal position, register (`.k`), 190
    - space, extra post-vertical (`\x`), 171
    - space, extra pre-vertical (`\x`), 171
    - spacing register (`.L`), 135
    - spacing, post-vertical (`pvs`), 171
    - thickness (`\D't ...'`), 194
    - vertical, drawing (`\L`), 191
  - line-tabs mode, 138
  - lines, blank, disabling, 135
  - lines
    - centering (`ce`), 17, 127
    - consecutive hyphenated (`hlm`), 130
    - interrupted, and input line traps (`itc`), 199
  - list, 19
    - of available glyphs (`goff_char(7)` man page), 158
  - ll request, using + and -, 110
  - location, vertical
    - page, marking (`mk`), 187
    - page, returning to marked (`rt`), 187
  - logical
    - not, limitation in expression, 109
    - operators, 109
  - long names, 223
  - loops and conditionals, 178
  - lq glyph, and lq string [`man`], 26
  - ls request, alternative to (`pvs`), 172
  - lt request, using + and -, 110
- m -
- M unit, 108
  - m unit, 108
  - machine unit (`u`), 108
  - macro
    - arguments, 114
    - arguments, and compatibility mode, 218
    - arguments, and tabs, 114
    - basics, 16
    - directories, 12
    - files, searching, 13
    - packages, 5, 21
    - packages, structuring the source code, 112
  - macro
    - appending (`am`), 184
    - arguments (`\$`), 185
    - creating alias (`als`), 177
    - end-of-input (`em`), 200
    - name register (`\$0`), 186
    - names, starting with [ `or` ], and `refer`, 111
    - removing (`rm`), 177
    - removing alias (`rm`), 177
    - renaming (`rn`), 177
  - macros, 115
    - recursive, 181
    - searching, 12
    - shared name space with strings and diversions, 174
    - tutorial for users, 16

- writing, [182](#)
- macros for manual pages [`man`], [22](#)
- magnification of a font (`fzoom`), [153](#)
- major quotes, [19](#)
- major version number register (`.x`), [123](#)
- man pages, [21](#)
- `man` macros, [22](#)
  - bold face, [24](#)
  - custom headers and footers, [27](#)
  - default indentation, [25](#)
  - empty space before a paragraph, [25](#)
  - hanging indentation, [24](#)
  - how to set fonts, [24](#)
  - italic fonts, [25](#)
  - line breaks with vertical space, [24](#)
  - line breaks without vertical space, [24](#)
  - moving left margin, [24](#)
  - resetting default indentation, [24](#)
  - tab stops, [25](#)
  - Ultrix-specific, [27](#)
- manipulating
  - filling and adjusting, [124](#)
  - hyphenation, [128](#)
  - spacing, [133](#)
- `man` macros, BSD compatibility, [25, 26](#)
- manual pages, [21](#)
- margin
  - bottom, [149](#)
  - top, [149](#)
- margin
  - for hyphenation (`hym`), [132](#)
  - glyph (`mc`), [216](#)
  - left (`po`), [145](#)
- mark, high-water, register (`.h`), [203](#)
- marking vertical page location (`mk`), [187](#)
- MathML, [293](#)
- maximum values of Roman numerals, [122](#)
- `mdoc` macros, [29](#)
- `me` macro package, [53](#)
- measurement unit, [108](#)
- measurements, [108](#)
  - specifying safely, [109](#)
- minimum values of Roman numerals, [122](#)
- minor version number register (`.y`), [123](#)
- minutes, current time (`minutes`), [123](#)
- `mm` macro package, [53](#)
- mode
  - compatibility, [223](#)
  - compatibility, and parameters, [218](#)
  - copy, [184](#)
  - copy-in, [184](#)
  - copy-in, and macro arguments, [185](#)
  - fill, [106, 126, 222](#)
  - line-tabs, [138](#)
  - nroff, [144](#)
  - safer, [10, 13, 122, 210, 212, 212, 213](#)
  - troff, [144](#)
  - unsafe, [11, 13, 122, 210, 212, 212, 213](#)
- mode
  - copy-in, and `\!`, [204](#)
  - copy-in, and `\?`, [178, 204](#)
  - copy-in, and `\a`, [138](#)
  - copy-in, and `cf` request, [210](#)
  - copy-in, and `device` request, [214](#)
  - copy-in, and `\E`, [141](#)
  - copy-in, and `ig` request, [118](#)
  - copy-in, and `length` request, [176](#)
  - copy-in, and `output` request, [205](#)
  - copy-in, and `\t`, [136](#)
  - copy-in, and `tm` request, [219](#)
  - copy-in, and `tml` request, [219](#)
  - copy-in, and `tmc` request, [219](#)
  - copy-in, and `trf` request, [210](#)
  - copy-in, and `\V`, [214](#)
  - copy-in, and `write` request, [213](#)
  - copy-in, and `writec` request, [213](#)
  - copy-in, and `writem` request, [213](#)
  - fill (`fi`), [124](#)
  - fill, and `\c`, [148](#)
  - for constant glyph space (`cs`), [166](#)
  - no-fill (`nf`), [124](#)
  - no-fill, and `\c`, [148](#)
  - no-space (`ns`), [135](#)
- modifying requests, [113](#)
- `mom` macro package, [83](#)
- month of the year register (`mo`), [123](#)
- motion operators, [110](#)
- motion
  - horizontal (`\h`), [189](#)
  - vertical (`\v`), [188](#)
- motions, page, [187](#)
- mounting font (`fp`), [155](#)
- `ms` macros, [29](#)
  - accent marks, [49](#)
  - body text, [35](#)
  - cover page, [34](#)
  - creating table of contents, [47](#)
  - displays, [43](#)
  - document control registers, [30](#)
  - equations, [44](#)
  - figures, [44](#)
  - footers, [46](#)
  - footnotes, [45](#)
  - general structure, [29](#)
  - headers, [46](#)
  - headings, [37](#)
  - highlighting, [38](#)
  - keeps, [43](#)
  - lists, [39](#)
  - margins, [47](#)
  - multiple columns, [47](#)
  - naming conventions, [53](#)
  - nested lists, [42](#)
  - page layout, [46](#)
  - paragraph handling, [35](#)
  - references, [44](#)
  - special characters, [49](#)

- strings, 49
  - tables, 44
  - ms macros, differences from AT&T, 51
  - multi-file documents, 219
  - multi-line strings, 174
  - multi-page table, example markup [ms], 45
  - multiple columns [ms], 47
- n -
- \n
    - and warnings, 222
    - unit, 108
    - when reading text for a macro, 184
  - \N
    - allowed delimiters, 116
    - and translations, 142
  - name space, common, of macros, diversions, and strings, 174
  - name
    - background color, register (.M), 210
    - drawing color, register (.m), 209
    - fill color, register (.M), 210
  - named character (\C), 159
  - names, long, 223
  - naming conventions, ms macros, 53
  - negating register values, 119
  - nested
    - assignments, 120
    - diversions, 202
  - nested lists [ms], 42
  - new page (bP), 17, 150
  - newline
    - character, 110, 116
    - character, and translations, 142
    - character, in strings, escaping, 174
    - final, stripping in diversions, 175
  - next
    - file, processing (nX), 211
    - free font position register (.fP), 156
  - ne request
    - and the .trunc register, 197
    - comparison with sv, 151
  - nf request, causing implicit linebreak, 124
  - nI register
    - and .d, 202
    - difference to .h, 203
  - nm request, using + and -, 110
  - no-break control
    - character (r), 112
    - character, changing (c2), 140
  - no-fill mode
    - (nF), 124
    - and \c, 148
  - no-space mode (nS), 135
  - node, output, 217
  - nr request, and warnings, 222
  - nroff mode, 144
  - nroff, the program, 3
  - nr request, using + and -, 110
  - number
    - input line, setting (lF), 219
    - of arguments register (.S), 185
    - of registers register (.R), 122
    - page (pN), 150
    - register, creating alias (aIn), 120
    - register, removing (rR), 119
    - register, renaming (rNn), 119
    - registers, dumping (pnr), 220
  - numbered
    - glyph (\N), 142, 160
    - list, example markup [ms], 40
  - numbers, and delimiters, 116
  - numbers, line, printing (nm), 214
  - numerals, Roman, 121
  - numeric expression, valid, 110
  - \n, incompatibilities with AT&T troff, 223
- o -
- \o, possible quote characters, 115
  - offset, page (pO), 145
  - open request, and safer mode, 10
  - opena request, and safer mode, 10
  - opening file (open), 213
  - operator, scaling, 110
  - operators
    - arithmetic, 109
    - as delimiters, 116
    - comparison, 109
    - logical, 109
    - motion, 110
    - unary, 109
  - operators, extremum (>?, <?), 110
  - optical size of a font, 153
  - options, 7
  - order of evaluation in expressions, 110
  - orientation, landscape, 14
  - orphan lines, preventing with ne, 150
  - os request, and no-space mode, 151
  - output
    - and input requests, 210
    - devices, 5, 287
    - encoding, ASCII, 10
    - encoding, cp1047, 11
    - encoding, EBCDIC, 11
    - encoding, latin-1 (ISO 8859-1), 10
    - encoding, utf-8, 10
    - intermediate, 294
    - node, 217
    - troff, 294
  - output
    - device name string register (.T), 11, 124
    - device usage number register (.T), 11
    - flush (f1), 220
    - glyphs, and input characters, compatibility with

AT&T `troff`, 224  
   `gtroff`, 294  
   line number register (`ln`), 123  
   line, continuation (`\c`), 148  
   line, horizontal position, register (`.k`), 190  
   request, and copy-in mode, 205  
   suppressing (`\o`), 207  
   transparent (`\!`, `\?`), 204  
   transparent (`cf`, `trf`), 210  
   transparent, incompatibilities with AT&T  
     `troff`, 225  
 output request, and `\!`, 205  
 overlapping characters, 161  
 overstriking glyphs (`\o`), 190

## - p -

P  
   unit, 108  
   used as delimiter, 116, 116  
 P unit, 108  
 packages, macros, 21  
 padding character, for fields (`fc`), 139  
 page  
   control, 150  
   footers, 196  
   headers, 196  
   layout, 148  
   location traps, 195  
   motions, 187  
   orientation, landscape, 14  
 page  
   break, conditional (`ne`), 150  
   ejecting register (`.pe`), 198  
   layout [`ms`], 46  
   length (`p1`), 149  
   length register (`.p`), 149  
   location, vertical, marking (`mk`), 187  
   location, vertical, returning to marked (`rt`),  
   187  
   new (`bp`), 150  
   number (`p1`), 150  
   number character (`%`), 149  
   number character, changing (`pc`), 150  
   number register (`%`), 150  
   offset (`po`), 145  
 paper  
   formats, 20  
   size, 14  
 paragraphs, 18  
 parameters, 185  
   and compatibility mode, 218  
 parentheses, 110  
 path, for  
   font files, 13  
   `tmac` files, 13  
 patterns for hyphenation (`hpf`), 131

PDF, embedding, 289  
`pi` request, and safer mode, 10  
`pic`, the program, 248  
 pica unit (`P`), 108  
 pile, glyph (`\b`), 195  
`pi` request, and `groff`, 212  
 planting a trap, 195  
 platform-specific directory, 13  
`p1` request, using + and -, 110  
 PNG image generation from PostScript, 307  
`pn` request, using + and -, 110  
 point sizes, fractional, 172, 224  
 point  
   size registers (`.s`, `.ps`), 170  
   size registers, last-requested (`.psr`, `.sr`),  
   172  
   sizes, changing (`ps`, `ls`), 169  
   unit (`p`), 108  
 polygon  
   drawing (`\D'p ...'`), 193  
   solid, drawing (`\D'P ...'`), 193  
 position  
   absolute, operator (`|`), 110  
   horizontal input line, saving (`\k`), 190  
   horizontal, in input line, register (`hp`), 190  
   horizontal, in output line, register (`.k`), 190  
   of lowest text line (`.h`), 203  
   vertical, current (`n1`), 151  
   vertical, in diversion, register (`.d`), 202  
 positions, font, 155  
 post-vertical line spacing, 171  
 post-vertical line  
   spacing register (`.pvs`), 172  
   spacing, changing (`pvs`), 172  
 postprocessor access, 214  
 postprocessors, 5  
 PostScript  
   bounding box, 217  
   embedding, 288  
   fonts, 153  
   PNG image generation, 307  
`po` request, using + and -, 110  
`preconv`, the program, 284  
 prefix, for commands, 12  
 preprocessor, calling convention, 26  
 preprocessors, 5, 226  
 previous  
   font (`ft`, `\f[]`, `\fP`), 152  
   line length (`.n`), 207  
 print current page register (`.P`), 9  
 printing  
   backslash (`\`, `\e`, `\E`, `\[rs]`), 116, 225  
   line numbers (`nm`), 214  
   to `stderr` (`tm`, `tm1`, `tmc`), 219  
   zero-width (`\z`, `\Z`), 190, 190  
 process ID of `gtroff` register (`$$`), 124  
 processing next file (`nx`), 211

## properties of

characters (*cflags*), 160glyphs (*cflags*), 160*ps* request

and constant glyph space mode, 166

with fractional type sizes, 172

*ps*o request, and safer mode, 10*ps* requestincompatibilities with AT&T *troff*, 224

using + and -, 110

*pvs* request, using + and -, 110

## - q -

## quotes

major, 19

trailing, 174

## - r -

\r, used as delimiter, 116

## \R

allowed delimiters, 116

and warnings, 222

radical glyph, and *cflags*, 161

ragged-left, 125

ragged-right, 125

*rc* request, and glyph definitions, 162

read-only register, changing format, 122

reading from standard input (*rd*), 211

recursive macros, 181

*refer*, the program, 270reference, *gtroff*, 105references [*ms*], 44*refer*, and macro names starting with [ or ], 111

## register

creating alias (*aln*), 120

format (\g), 122

removing (*rr*), 119renaming (*rnn*), 119

## registers, 118

built-in, 122

## registers

interpolating (\n), 120

number of, register (.R), 122

setting (*nr*, \R), 118specific to *grohtml*, 293

## removing

alias, for diversion (*rm*), 177alias, for macro (*rm*), 177alias, for string (*rm*), 177diversion (*rm*), 177glyph definition (*rchar*, *rfschar*), 163macro (*rm*), 177number register (*rr*), 119request (*rm*), 177string (*rm*), 177

## renaming

diversion (*rn*), 177macro (*rn*), 177number register (*rnn*), 119request (*rn*), 177string (*rn*), 177

## request

arguments, 114

arguments, and compatibility mode, 218

undefined, 117

## request

removing (*rm*), 177renaming (*rn*), 177

## requests, 112

for drawing, 191

for input and output, 210

modifying, 113

## resolution

device, 308

horizontal, 307

vertical, 309

## resolution

horizontal, register (.H), 122

vertical, register (.V), 123

\RET, when reading text for a macro, 184

returning to marked vertical page location (*rt*), 187

revision number register (.Y), 124

*rf*, the program, 2right-justifying (*rj*), 128*rj* request, causing implicit linebreak, 124*rn* glyph, and *cflags*, 161*roff*, the program, 2

## roman glyph

correction after italic glyph (\/), 167

correction before italic glyph (\,), 168

## Roman numerals, 121

maximum and minimum, 122

*rq* glyph, at end of sentence, 106, 161*rq* glyph, and *rq* string [*man*], 26*rt* request, using + and -, 110

RUNOFF, the program, 2

*ru* glyph, and *cflags*, 161

## \R

after \c, 148

difference to *nr*, 120

using + and -, 110

## - s -

\S, allowed delimiters, 116

## \s

allowed delimiters, 116

unit, 108, 172

with fractional type sizes, 172

safer mode, 10, 13, 122, 210, 212, 212, 213

saving horizontal input line position (\k), 190

- scaling
  - indicator, 108
  - operator, 110
- searching
  - fonts, 13
  - macro files, 13
  - macros, 12
- seconds, current time (`seconds`), 123
- sentence space, 105
- sentence space size register (`.sss`), 126
- sentences, 105
- setting
  - diversion trap (`dt`), 198
  - end-of-input trap (`em`), 200
  - input line number (`lf`), 219
  - input line trap (`it`), 198
  - registers (`nr`, `\R`), 118
- shading filled objects (`\D'f ...'`), 193
- `shc` request, and translations, 142
- site-specific directory, 13, 13
- size
  - of type, 169
  - optical, of a font, 153
  - paper, 14
- size of
  - sentence space register (`.sss`), 126
  - word space register (`.ss`), 126
- sizes, 169
  - fractional, 172, 224
- skew, of last glyph (`.csk`), 207
- slant, font, changing (`\S`), 165
- `soelim`, the program, 282
- soft hyphen
  - character, setting (`shc`), 133
  - glyph (`hy`), 133
- solid
  - circle, drawing (`\D'C ...'`), 192
  - ellipse, drawing (`\D'E ...'`), 193
  - polygon, drawing (`\D'P ...'`), 193
- `sp` request
  - and no-space mode, 135
  - and traps, 134
  - causing implicit linebreak, 124
- space
  - between sentences, 105
  - character, 116
  - characters, in expressions, 110
  - discardable, horizontal, 127
  - discarded, in traps, 134
  - horizontal, unformatting, 175
  - unbreakable, 189
- space
  - between sentences register (`.sss`), 126
  - between words register (`.ss`), 126
  - character, zero width (`\&`), 113, 167, 191
  - horizontal (`\h`), 189
  - vertical, unit (`v`), 108
  - width of a digit (`\0`), 189
- spaces
  - in a macro argument, 114
  - leading and trailing, 105
- spaces with `ds`, 174
- spacing, 17
  - manipulating, 133
  - vertical, 169
- special
  - characters, 142, 287
  - fonts, 157, 164, 309
  - fonts, emboldening, 166
- special
  - characters [`ms`], 49
  - request, and font translations, 153
  - request, and glyph search order, 157
- spline, drawing (`\D' ...'`), 193
- springing a trap, 195
- `sqrtext` glyph, and `cflags`, 161
- stacking glyphs (`\b`), 195
- standard input, reading from (`rd`), 211
- `stderr`, printing to (`tm`, `tml`, `tmc`), 219
- stops, tabulator, 106
- string
  - arguments, 173
  - comparison, 178
- string
  - appending (`as`), 176
  - creating alias (`als`), 177
  - expansion (`\*`), 173
  - interpolation (`\*`), 173
  - length of (`length`), 176
  - removing (`rm`), 177
  - removing alias (`rm`), 177
  - renaming (`rn`), 177
- strings, 173
  - multi-line, 174
  - shared name space with macros and diversions, 174
- strings
  - [`ms`], 49
  - specific to `grohtml`, 293
- stripping final newline in diversions, 175
- structuring source code of documents or macro packages, 112
- `sty` request
  - and changing fonts, 152
  - and font positions, 156
  - and font translations, 153
- styles, font, 153
- substring (`substring`), 176
- suppressing output (`\0`), 207
- `sv` request, and no-space mode, 151
- switching environments (`ev`), 206
- `sy` request, and safer mode, 10
- symbol, 157
- symbol
  - defining (`char`), 162
  - table, dumping (`pm`), 220

symbols, using, 156

`system()` return value register (`systat`), 213

`\S`, incompatibilities with AT&T `troff`, 224

`\s`

incompatibilities with AT&T `troff`, 224

using + and -, 110

- t -

`\t`

and copy-in mode, 136

and translations, 142

and warnings, 222

used as delimiter, 116

`tab`

character, 106, 116

character, and translations, 142

line-tabs mode, 138

stops, 106

stops, for TTY output devices, 137

`tab`

character, non-interpreted (`\t`), 136

repetition character (`tc`), 138

settings register (`.tabs`), 137

stops [`man`], 25

table of contents, 20, 139

table of contents, creating [`ms`], 47

tables [`ms`], 44

tabs

and fields, 136

and macro arguments, 114

before comments, 117

`tbl`, the program, 237

Teletype, 287

terminal

conditional output for, 178

control sequences, 287

text, justifying, 124

text

`gtroff` processing, 105

justifying (`rj`), 128

line, position of lowest (`.h`), 203

thickness of lines (`\D't ...'`), 194

three-part title (`t1`), 149

`ti` request, causing implicit linebreak, 124

time, current, 212

time, current

hours (`hours`), 123

minutes (`minutes`), 123

seconds (`seconds`), 123

title page, example markup, 35

title line

(`t1`), 149

length register (`.lt`), 149

length (`lt`), 149

titles, 149

`ti` request, using + and -, 110

`tkf` request

and font styles, 154

and font translations, 153

with fractional type sizes, 172

`t1` request, and `mc`, 216

`tm` request, and copy-in mode, 219

`tm1` request, and copy-in mode, 219

`tmac`

directory, 13

path, 13

`tmc` request, and copy-in mode, 219

token, input, 217

top margin, 149

top-level diversion, 201

top-level diversion

and `\!`, 204

and `\?`, 205

and `bp`, 150

`tr` request

and glyph definitions, 162

and soft hyphen character, 133

track kerning, 167

track kerning, activating (`tkf`), 167

trailing

quotes, 174

spaces, 105

translations of characters, 140

transparent characters, 106, 161

transparent output

(`\!`, `\?`), 204

(`cf`, `trf`), 210

incompatibilities with AT&T `troff`, 225

`trap`

planting, 195

springing, 195

`trap`

changing location (`ch`), 197

distance, register (`.t`), 197

diversion, setting (`dt`), 198

end-of-input, setting (`em`), 200

input line, setting (`it`), 198

traps, 195

and discarded space, 134

and diversions, 198

blank line, 199

diversion, 198

end-of-input, 200

input line, 198

leading spaces, 199

page location, 195

traps

dumping (`ptr`), 220

input line, and interrupted lines (`itc`), 199

sprung by `bp` request (`.pe`), 198

`trf` request

and copy-in mode, 210

and invalid characters, 211

causing implicit linebreak, 124

`trin` request, and `asciify`, 205

`troff`

mode, 144

output, 294

truncated vertical space register (`.trunc`), 197

`tr` request, incompatibilities with AT&T `troff`, 224

TTY, conditional output for, 178

tutorial for macro users, 16

type

size, 169

sizes, fractional, 172, 224

type

size registers (`.s`, `.ps`), 170

sizes, changing (`ps`, `\s`), 169

- u -

u

unit, 108

used as delimiter, 116

`uf` request, and font styles, 154

`ul` request, and font translations, 153

Ultrix-specific `man` macros, 27

`ul` glyph, and `cflags`, 161

unary operators, 109

unbreakable space, 189

undefined

identifiers, 111

request, 117

underline font (`uf`), 165

underlining

continuous (`cu`), 165

(`ul`), 165

underscore glyph (`\[ru]`), 191

unformatting horizontal space, 175

unformatting diversions (`asciify`), 205

Unicode, 110, 160

unit

c, 108

f, 108

f, and colors, 209

i, 108

m, 108

M, 108

n, 108

P, 108

p, 108

s, 108, 172

u, 108

v, 108

z, 108, 172

units

default, 108

of measurement, 108

unnamed glyphs, 160

unnamed

fill colors (`\D'F...'`), 194

glyphs, accessing with `\N`, 310

unsafe mode, 11, 13, 122, 210, 212, 212, 213

\

used as delimiter, 116

as delimiter, 116, 116

as delimiter, 116

user's

macro tutorial, 16

tutorial for macros, 16

using symbols, 156

utf-8, output encoding, 10

- v -

`\v`, and copy-in mode, 214

`\v`

allowed delimiters, 116

internal representation, 218

unit, 108

valid numeric expression, 110

value, incrementing without changing the register, 121

variables in environment, 11

version number

major, register (`.x`), 123

minor, register (`.y`), 123

vertical

line spacing, effective value, 171

resolution, 309

spacing, 169

vertical

line drawing (`\L`), 191

line spacing register (`.v`), 171

line spacing, changing (`vs`), 171

motion (`\v`), 188

page location, marking (`mk`), 187

page location, returning to marked (`rt`), 187

position in diversion register (`.d`), 202

position trap enable register (`.vpt`), 195

position traps, enabling (`vpt`), 195

position, current (`nl`), 151

resolution register (`.v`), 123

space unit (`v`), 108

- w -

`\w`, allowed delimiters, 116

warnings, 221, 221

warnings, level (`warn`), 221

what is `groff`?, 2

`\`, when reading text for a macro, 184

while, 180

`while` request

and font translations, 153

operators to use with, 178

`while` request

and the `'!` operator, 109

confusing with `br`, 181

whitespace characters, [110](#)

width

escape (`\w`), [189](#)

of last glyph (`.w`), [207](#)

word space size register (`.ss`), [126](#)

`write` request, and copy-in mode, [213](#)

`writec` request, and copy-in mode, [213](#)

`writem` request, and copy-in mode, [213](#)

writing macros, [182](#)

writing to file (`write`, `writec`), [213](#)

- x -

`\X`

and special characters, [214](#)

possible quote characters, [115](#)

`\x`, allowed delimiters, [116](#)

`\X`, followed by `\%`, [131](#)

- y -

year, current, register (`year`, `yr`), [123](#)

`\Y`, followed by `\%`, [131](#)

- z -

z unit, [108](#), [172](#)

`\Z`, allowed delimiters, [116](#)

zero width space character (`\&`), [113](#), [167](#), [191](#)

zero-width printing (`\z`, `\Z`), [190](#), [190](#)

zoom factor of a font (`fzoom`), [153](#)

- | -

|

and page motion, [110](#)

used as delimiter, [116](#)

`\|`, incompatibilities with AT&T `troff`, [224](#)

- } -

\

, difference to `\SP`, [114](#)

disabling (`eo`), [140](#)