groff

The GNU implementation of `troff`
Edition 1.19.2
Summer 2005

by Trent A. Fisher
and Werner Lemberg (`bug-groff@gnu.org`)

This manual documents GNU `troff` version 1.19.2.

Table of Contents

1 Introduction

GNU `troff` (or `groff`) is a system for typesetting documents. `troff` is very flexible and has been in existence (and use) for about 3 decades. It is quite widespread and firmly 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 WYSIWYG[1] 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.

Likewise, `groff` should not be confused with a *word processor*, since that term connotes an integrated system that includes an editor and a 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 available 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 J. E. Saltzer, which ran on MIT's CTSS operating system in the mid-sixties. This name came from the common phrase of the time "I'll run off a document." 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 McIllroy rewrote an extended and simplified version of `roff` in the BCPL programming language.

---

[1] What You See Is What You Get

The first version of UNIX was developed on a PDP-7 which was sitting around Bell Labs. In 1971 the developers wanted to get a PDP-11 for further work on the operating system. In order to justify the cost for this system, they proposed that they would implement a document formatting system for the AT&T patents division. This first formatting program was a reimplementation of McIllroy'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 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 which were not applicable (e.g. font changes).

Since there are several things which 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 formulæ 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 was killed in a car accident.

So, Brian Kernighan took on the task of rewriting `troff`. The newly rewritten version produced device independent code which 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 which would produce ASCII output.
- A version of the '`me`' macros and an implementation of the '`man`' macros.

Also, a front-end was included which 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 output 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 which 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`), and for processing bibliographies (`refer`). An associated program which 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.

There are other preprocessors in existence, but, unfortunately, no free implementations are available. Among them are preprocessors for drawing mathematical pictures (`ideal`) and chemical structures (`chem`).

1.6 Output Devices

`groff` actually produces device independent code which 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), and HTML.

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 Section 2.2 [Environment], page 12, 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 AT&T troff which is the common origin of all troff derivates (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, grefer, or gsoelim.

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 Section 6.3.1 [Invoking gpic], page 175, Section 6.1.1 [Invoking geqn], page 175, Section 6.2.1 [Invoking gtbl], page 175, Section 6.4.1 [Invoking ggrn], page 175, Section 6.6.1 [Invoking grefer], page 175, Section 6.7.1 [Invoking gsoelim], page 175, Section 7.2.1 [Invoking grotty], page 177, Section 7.3.1 [Invoking grops], page 177, Section 7.7.1 [Invoking grohtml], page 177, Section 7.4.1 [Invoking grodvi], page 177, Section 7.5.1 [Invoking grolj4], page 177, Section 7.6.1 [Invoking grolbp], page 177, and Section 7.8.1 [Invoking gxditview], page 178.

The command line format for groff is:

```
groff [ -abceghilpstvzCEGNRSUVXZ ] [ -Fdir ] [ -mname ]
      [ -Tdef ] [ -ffam ] [ -wname ] [ -Wname ]
      [ -Mdir ] [ -dcs ] [ -rcn ] [ -nnum ]
      [ -olist ] [ -Parg ] [ -Larg ] [ -Idir ]
      [ files... ]
```

The command line format for gtroff is as follows.

```
gtroff [ -abcivzCERU ] [ -wname ] [ -Wname ] [ -dcs ]
       [ -ffam ] [ -mname ] [ -nnum ]
       [ -olist ] [ -rcn ] [ -Tname ]
       [ -Fdir ] [ -Mdir ] [ 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:

'`-h`' Print a help message.

'`-e`' Preprocess with `geqn`.

'`-t`' Preprocess with `gtbl`.

'`-g`' Preprocess with `ggrn`.

'`-G`' Preprocess with `grap`.

'`-p`' Preprocess with `gpic`.

'`-s`' Preprocess with `gsoelim`.

'`-c`' Suppress color output.

'`-R`' Preprocess with `grefer`. No mechanism is provided for passing arguments to `grefer` because most `grefer` options have equivalent commands which can be included in the file. See Section 6.6 [grefer], page 175, 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.

'`-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.

'`-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.

'`-P`*arg*' 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.

'`-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 Section 8.2 [Font Files], page 192, for more info). If not present, '`-l`' is ignored.

‘`-L`*arg*’ 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.

‘`-T`*dev*’ 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.

`dvi` For TeX 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 which use the Unicode (ISO 10646) character set with UTF-8 encoding.

`cp1047` For typewriter-like devices which 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` To produce HTML output. Note that the HTML 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 Section 5.6.5 [Built-in Registers], page 74.

The postprocessor to be used for a device is specified by the `postpro` command in the device description file. (See Section 8.2 [Font Files], page 192, for more info.) This can be overridden with the ‘`-X`’ option.

'-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.

'-N' Don't allow newlines with `eqn` delimiters. This is the same as the '-N' option in `geqn`.

'-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.

'-U' Unsafe mode. This enables the `open`, `opena`, `pso`, `sy`, and `pi` requests.

'-a' Generate an ASCII approximation of the typeset output. The read-only register `.A` is then set to 1. See Section 5.6.5 [Built-in Registers], page 74. 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.

'-i' Read the standard input after all the named input files have been processed.

'-w*name*' Enable warning *name*. Available warnings are described in Section 5.33 [Debugging], page 167. Multiple '-w' options are allowed.

'-W*name*' Inhibit warning *name*. Multiple '-W' options are allowed.

'-E' Inhibit all error messages.

'-C' Enable compatibility mode. See Section 5.34 [Implementation Differences], page 171, for the list of incompatibilities between `groff` and AT&T `troff`.

'-d*cs*'
'-d*name*=*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).

'-f*fam*' Use *fam* as the default font family. See Section 5.17.2 [Font Families], page 104.

'-m*name*' 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).

'-n*num*' Number the first page *num*.

'-o*list*' 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 Section 5.6.5 [Built-in Registers], page 74.

If your document restarts page numbering at the beginning of each chapter, then gtroff prints the specified page range for each chapter.

'-r*cn*'
'-r*name*=*n*'
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).

'-F*dir*' Search '*dir*' for subdirectories 'dev*name*' (*name* is the name of the device), for the 'DESC' file, and for font files before looking in the standard directories (see Section 2.4 [Font Directories], page 13). This option is passed to all pre- and postprocessors using the GROFF_FONT_PATH environment variable.

'-M*dir*' Search directory '*dir*' for macro files before the standard directories (see Section 2.3 [Macro Directories], page 12).

'-I*dir*' This option may be used to specify a directory to search for files. It is passed to the following programs:

- gsoelim (see Section 6.7 [gsoelim], page 175 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 will be searched in the order specified. No directory search is performed for files specified using an absolute path.

2.2 Environment

There are also several environment variables (of the operating system, not within `gtroff`) which can modify the behavior of `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`, `refer`, and `soelim`. It does not apply to `grops`, `grodvi`, `grotty`, `pre-grohtml`, `post-grohtml`, `grolj4`, 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_TMAC_PATH`
: A colon-separated list of directories in which to search for macro files (before the default directories are tried). See Section 2.3 [Macro Directories], page 12.

`GROFF_TYPESETTER`
: The default output device.

`GROFF_FONT_PATH`
: A colon-separated list of directories in which to search for the `dev`*name* directory (before the default directories are tried). See Section 2.4 [Font Directories], page 13.

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

`GROFF_TMPDIR`
: The directory in which `groff` creates temporary files. If this is not set and `TMPDIR` 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.

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 '`-m`*`name`*' command line option work. The `mso` 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/1.18.2/tmac
  ```

  assuming that the version of groff is 1.18.2, 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 Section 5.17.2 [Font Families], page 104), 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 will always append the directory dev*name*, 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/1.18.2/font
  ```

  assuming that the version of groff is 1.18.2, 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 Section 5.15 [Page Layout], page 100, for vertical manipulation of the page size. See Section 5.13 [Line Layout], page 95, 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 Section 8.2.1 [DESC File Format], page 192, 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.[1]

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

[1] 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.

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 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.[1] 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 Section 5.5 [Embedded Commands], page 64.

The word *argument* is used in this chapter to mean a word or number which 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 Section 5.5.1.1 [Request and Macro Arguments], page 65.

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

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

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 which 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.

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`’.[2]

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 *N*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 Section 5.2 [Measurements], page 60).

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:

[2] If you need finer granularity of the vertical space, use the `pvs` request (see Section 5.18.1 [Changing Type Sizes], page 120).

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

The '`.ce 0`' request tells `groff` 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 which are done in all documents. These common operations are written into *macros* and collected into a *macro package*.

All macro packages provide certain common capabilities which 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 Section 5.15 [Page Layout], page 100, 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 which 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 which 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 which 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 which 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 $-0.5\,$i.

`-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 14.

`-rIN=`*`length`*
: Set the body text indentation to *length*. If not specified, the indentation defaults to 7 n (7 characters) in nroff mode and 7.2 n

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 78 n in nroff mode (this is 78 characters per line) and 6.5 i in troff mode.[1]

`-rLT=`*`length`*
: Set title length to *length*. If not specified, the title length defaults to the line length.

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

`-rS`*`xx`*
: 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 3 n.

`-rX`*`nnn`*
: After page *nnn*, number pages as *nnn*a, *nnn*b, *nnn*c, 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*]]] [Macro]

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*

---

[1] 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 which evaluates equal to 65 n, i.e., the formatter's default line length in nroff mode), will *not* set the line length to 65 n (it will be adjusted to the `man` macro package's default setting of 78 n), whereas the use of the '`-rLL=65n`' option, or the '`.nr LL 65n`' request *will* establish a line length of 65 n.

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 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*] [Macro]

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 following text is reset to its default value.

`.SS` [*heading*] [Macro]

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*] [Macro]

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` [Macro]
`.PP` [Macro]
`.P` [Macro]

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*]] [Macro]

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*] [Macro]

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*] [Macro]

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*] [Macro]

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 point. If command line option '`-rS=`*n*' is given, use *n* pt as the default text size.

`.SM` [*text*] [Macro]

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*] [Macro]

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* [Macro]

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* [Macro]

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

`.RI` *text* [Macro]

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

`.IR` *text* [Macro]

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

`.BR` *text* [Macro]

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

`.RB` *text* [Macro]

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

`.B` [*text*] [Macro]

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*] [Macro]

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

`.DT` [Macro]

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*] [Macro]

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, 0.4 v 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*]] [Macro]

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*] [Macro]

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]` [String]

Switch back to the default font size.

`\*[HF]` [String]

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

`\*[R]` [String]

The 'registered' sign.

`\*[Tm]` [String]
The 'trademark' sign.

`\*[lq]` [String]
`\*[rq]` [String]
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` [Macro]
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` [Macro]
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 −0.5 i.

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* [Macro]
Print '`<CTRL/key>`'.

`.CW` [Macro]
Print subsequent text using the constant width (Courier) typeface.

`.Ds` [Macro]
Begin a non-filled display.

`.De` [Macro]
End a non-filled display started with `Ds`.

`.EX` [*indent*] [Macro]
Begins a non-filled display using the constant width (Courier) typeface. Use the optional *indent* argument to indent the display.

`.EE` [Macro]
End a non-filled display started with `EX`.

`.G` [*text*] [Macro]
Sets *text* in Helvetica. If no text is present on the line where the macro is called, then the text of the next line appears in Helvetica.

`.GL` [*text*] [Macro]
Sets *text* in Helvetica Oblique. If no text is present on the line where the macro is called, then the text of the next line appears in Helvetica Oblique.

`.HB` [*text*] [Macro]
Sets *text* in Helvetica Bold. If no text is present on the line where the macro is called, then all text up to the next `HB` appears in Helvetica Bold.

`.TB` [*text*] [Macro]
Identical to `HB`.

`.MS` *title sect* [*punct*] [Macro]
Set a manpage reference in Ultrix format. The *title* is in Courier instead of italic. Optional punctuation follows the section number without an intervening space.

`.NT` [`C`] [*title*] [Macro]
Begin a note. Print the optional *title*, or the word "Note", centered on the page. Text following the macro makes up the body of the note, and is indented on both sides. If the first argument is `C`, the body of the note is printed centered (the second argument replaces the word "Note" if specified).

`.NE` [Macro]
End a note begun with `NT`.

`.PN` *path* [*punct*] [Macro]
Set the path name in constant width (Courier), followed by optional punctuation.

`.Pn` [*punct*] *path* [*punct*] [Macro]
When called with two arguments, identical to `PN`. When called with three arguments, set the second argument in constant width (Courier), bracketed by the first and third arguments in the current font.

`.R` [Macro]
Switch to roman font and turn off any underlining in effect.

`.RN` [Macro]
Print the string '<RETURN>'.

`.VS [4]` [Macro]
Start printing a change bar in the margin if the number 4 is specified. Otherwise, this macro does nothing.

`.VE` [Macro]
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 Section 4.3.7 [Differences from AT&T ms], page 54, 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 Section 4.3.3 [ms Document Control Registers], page 33, for more details.

Cover page
: A cover page consists of a title, the author's name and institution, an abstract, and the date.[2] See Section 4.3.4 [ms Cover Page Macros], page 36, 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 Section 4.3.5 [ms Body Text], page 38, 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.

---

[2] Actually, only the title is required.

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]` [Register]

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]` [Register]

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

Effective: next paragraph.

Default: 6 i.

`\n[LT]` [Register]

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

Effective: next paragraph.

Default: 6 i.

`\n[HM]` [Register]

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

Effective: next page.

Default: 1 i.

`\n[FM]` [Register]

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

Effective: next page.

Default: 1 i.

Text Settings

`\n[PS]` [Register]

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]` [Register]

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]` [Register]

Defines an increment in point size, which will be 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]` [Register]

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` will be printed at the point size set by `PS`; for each level below the value of `GROWPS`, the point size will be increased in steps equal to the value of `PSINCR`. Setting `GROWPS` to any value less than 2 disables the incremental heading size feature.

Effective: next section heading.

Default: 0.

`\n[HY]` [Register]

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: 14.

`\n[FAM]` [Register]

Defines the font family used to typeset the document.

Effective: next paragraph.

Default: as defined in the output device.

Paragraph Settings

`\n[PI]` [Register]

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

Effective: next paragraph.

Default: 5 n.

`\n[PD]` [Register]

Defines the space between paragraphs.

Effective: next paragraph.

Default: 0.3 v.

`\n[QI]` [Register]

Defines the indentation on both sides of a quoted (`QP` macro) paragraph.

Effective: next paragraph.

Default: 5 n.

`\n[PORPHANS]` [Register]

Defines the minimum number of initial lines of any paragraph which 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 will be forced, before the start of the paragraph.

Effective: next paragraph.

Default: 1.

`\n[HORPHANS]` [Register]

Defines the minimum number of lines of the following paragraph which 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 will be forced before the heading.

Effective: next paragraph.

Default: 1.

Footnote Settings

`\n[FL]` [Register]

Defines the length of a footnote.

Effective: next footnote.

Default: `\n[LL]` ∗ 5/6.

`\n[FI]` [Register]

Defines the footnote indentation.

Effective: next footnote.

Default: 2 n.

`\n[FF]` [Register]

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.

`\n[FPS]` [Register]

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: `\n[PS]` − 2.

`\n[FVS]` [Register]

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: `\n[FPS]` + 2.

`\n[FPD]` [Register]

Defines the footnote paragraph spacing.

Effective: next footnote.

Default: `\n[PD]`/2.

Miscellaneous Number Registers

`\n[MINGW]` [Register]

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

Effective: next page.

Default: 2 n.

4.3.4 Cover page macros

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

`.RP [no]` [Macro]

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

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

`.AU` [Macro]
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` [Macro]
Specifies the author's institution. You can specify multiple institutions in the same way that you specify multiple authors.

`.AB [no]` [Macro]
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` [Macro]
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` [Macro]
`.LP` [Macro]

Sets a paragraph with an initial indentation.

`.QP` [Macro]

Sets a paragraph that is indented at both left and right margins. The effect is identical to the HTML `<BLOCKQUOTE>` element. The next paragraph or heading returns margins to normal.

`.XP` [Macro]

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 Section 4.3.3 [ms Document Control Registers], page 33) 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* [Macro]
`.NH S` *level0 ...* [Macro]

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]` [String]
`\*[SN-DOT]` [String]
`\*[SN-NO-DOT]` [String]

After invocation of `NH`, the assigned section number is made available in the strings `SN-DOT` (exactly as it appears in the printed section heading) and `SN-NO-DOT` (with the final 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

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

before your first use of `NH`, or simply

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

after your first use of `NH`.

`.SH` [*match-level*] [Macro]

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 Section 4.3.3 [ms Document Control Registers], page 33.

The `HORPHANS` register (see Section 4.3.3 [ms Document Control Registers], page 33) 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*]]] [Macro]

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

`.I` [*txt* [*post* [*pre*]]] [Macro]
Sets its first argument in *italic type*. It operates similarly to the `B` macro otherwise.

`.CW` [*txt* [*post* [*pre*]]] [Macro]
Sets its first argument in a `constant width face`. It operates similarly to the `B` macro otherwise.

`.BI` [*txt* [*post* [*pre*]]] [Macro]
Sets its first argument in bold italic type. It operates similarly to the `B` macro otherwise.

`.BX` [*txt*] [Macro]
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*]] [Macro]
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` [Macro]
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` [Macro]
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` [Macro]
Prints all text following in the normal point size (that is, the value of the `PS` register).

`\*[{]` [String]
`\*[}]` [String]
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*]] [Macro]

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 Section 4.3.3 [ms Document Control Registers], page 33) 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 Section 4.3.5.5 [Indentation values in ms], page 45, 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 indentation a section of text while still wrapping and filling. See Section 4.3.5.4 [Lists in ms], page 42, for an example of nested lists.

`.RS` [Macro]
`.RE` [Macro]

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 Section 4.3.5.7 [ms Displays and Keeps], page 45, 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 Section 5.10 [Tabs and Fields], page 86.

`.TA` [Macro]

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` [Macro]
`.LD` [Macro]
`.DE` [Macro]

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

`.DS I` [Macro]
`.ID` [Macro]
`.DE` [Macro]

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. The `ID` macro allows the display to break across pages. The `DE` macro ends the display.

`.DS B` [Macro]
`.BD` [Macro]
`.DE` [Macro]

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. The `BD` macro allows the display to break across pages. The `DE` macro ends the display.

`.DS C` [Macro]
`.CD` [Macro]
`.DE` [Macro]

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. The `CD` macro allows the display to break across pages. The `DE` macro ends the display.

`.DS R` [Macro]
`.RD` [Macro]
`.DE` [Macro]

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. The `RD` macro allows the display to break across pages. The `DE` macro ends the display.

`.Ds` [Macro]
`.De` [Macro]

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 which actually use `Ds` and `De` always load a specific macro file from the X11 distribution ('`macros.t`') which 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. The '`ms`' macros provide the `KS` and `KE` macros for this purpose.

`.KS` [Macro]

`.KE` [Macro]

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

`.KF` [Macro]
`.KE` [Macro]

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` [Macro]
`.B2` [Macro]

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] [Macro]
`.TE` [Macro]

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` [Macro]
`.PE` [Macro]

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*] [Macro]

`.EN` [Macro]

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.

`.[` [Macro]
`.]` [Macro]

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).

`\*[*]` [String]

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

`.FS` [Macro]
`.FE` [Macro]

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 Section 4.3.3 [ms Document Control Registers], page 33.

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]` [String]
`\*[CH]` [String]
`\*[RH]` [String]

Sets the left, center, and right headers.

`\*[LF]` [String]
`\*[CF]` [String]
`\*[RF]` [String]

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*`'` [Macro]
`.EH` `'`*left*`'`*center*`'`*right*`'` [Macro]
`.OF` `'`*left*`'`*center*`'`*right*`'` [Macro]
`.EF` `'`*left*`'`*center*`'`*right*`'` [Macro]

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.

4.3.6.2 Margins

You control margins using a set of number registers. See Section 4.3.3 [ms Document Control Registers], page 33, for details.

4.3.6.3 Multiple columns

The '`ms`' macros can set text in as many columns as will 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` [Macro]
Single-column mode.

`.2C` [Macro]
Two-column mode.

`.MC` [*width* [*gutter*]] [Macro]
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` number 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*] [Macro]
`.XA` [*page*] [Macro]
`.XE` [Macro]
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`] [Macro]

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`] [Macro]

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]` [String]

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

`\*[ABSTRACT]` [String]

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

`\*[TOC]` [String]

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

`\*[MONTH1]` [String]
`\*[MONTH2]` [String]
`\*[MONTH3]` [String]
`\*[MONTH4]` [String]
`\*[MONTH5]` [String]
`\*[MONTH6]` [String]
`\*[MONTH7]` [String]
`\*[MONTH8]` [String]
`\*[MONTH9]` [String]
`\*[MONTH10]` [String]
`\*[MONTH11]` [String]
`\*[MONTH12]` [String]

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

The following special characters are available[3]:

`\*[-]` [String]

Prints an em dash.

`\*[Q]` [String]
`\*[U]` [String]

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` [Macro]

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:

`\*[']` [String]

Acute accent.

[3] For an explanation what special characters are see Section 7.1 [Special Characters], page 177.

*[`] [String]
Grave accent.

*[^] [String]
Circumflex.

*[,] [String]
Cedilla.

*[~] [String]
Tilde.

*[:] [String]
Umlaut.

*[v] [String]
Hacek.

*[_] [String]
Macron (overbar).

*[.] [String]
Underdot.

*[o] [String]
Ring above.

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

*[?] [String]
Upside-down question mark.

*[!] [String]
Upside-down exclamation point.

*[8] [String]
German ß ligature.

*[3] [String]
Yogh.

*[Th] [String]
Uppercase thorn.

*[th] [String]
Lowercase thorn.

*[D-] [String]
Uppercase eth.

`\*[d-]` [String]
Lowercase eth.

`\*[q]` [String]
Hooked o.

`\*[ae]` [String]
Lowercase æ ligature.

`\*[Ae]` [String]
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 (this 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]` [Register]
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.

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:

`.TM` Technical memorandum; a cover sheet style

`.IM` Internal memorandum; a cover sheet style

`.MR` Memo for record; a cover sheet style

`.MF` Memo for file; a cover sheet style

`.EG` Engineer's notes; a cover sheet style

`.TR` Computing Science Tech Report; a cover sheet style

`.OK` Other keywords

`.CS` Cover sheet information

`.MH` 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.

`.AM` [Macro]
: Improved accent marks. See Section 4.3.6.5 [ms Strings and Special Characters], page 51, for details.

`.DS I` [Macro]
: Indented display. The default behavior of AT&T `troff -ms` was to indent; the `groff` default prints displays flush left with the body text.

`.CW` [Macro]
: Print text in `constant width` (Courier) font.

`.IX` [Macro]
: 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]` [Register]
: 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 Section 4.3.6.5 [ms Strings and Special Characters], page 51, 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 '`meref.me`' documents in groff's '`doc`' directory.

4.5 'mm'

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

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 Section 5.1.5 [Implicit Line Breaks], page 58).

See Section 5.7 [Manipulating Filling and Adjusting], page 75.

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 Section 5.8 [Manipulating Hyphenation], page 79.

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 typsetting 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 Section 5.1.6 [Input Conventions], page 59.)

In addition, the following characters and symbols are treated transparently while handling end-of-sentence characters: ‘`"`’, ‘`'`’, ‘`)`’, ‘`]`’, ‘`*`’, `\[dg]`, and `\[rq]`.

See the `cflags` request in Section 5.17.4 [Using Symbols], page 108, 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 Section 5.10 [Tabs and Fields], page 86.

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 Section 5.24.4 [Blank Line Traps], page 148.

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.

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 Section 5.7 [Manipulating Filling and Adjusting], page 75.

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-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 ...
```

will fail if you use the Euro character in the input. Usually, this limitation is present only for devices which have a limited set of output glyphs (e.g. '`-Tascii`' and '`-Tlatin1`'); for other devices it is usually sufficient to install proper fonts which 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. With 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

gtroff (like many other programs) requires numeric parameters to specify various measurements. Most numeric parameters[1] may have a *measurement unit* attached. These units are specified as a single character which immediately follows the number or expression. Each of these units are understood, by gtroff, to be a multiple of its *basic unit*. So, whenever a different measurement unit is specified gtroff converts this into its *basic units*. This basic unit, represented by a 'u', is a device dependent measurement which is quite small, ranging from 1/75 th to 1/72000 th of an inch. The values may be given as fractional numbers; however, fractional basic units are always rounded to integers.

Some of the measurement units are completely independent of any of the current settings (e.g. type size) of gtroff.

i
: Inches. An antiquated measurement unit still in use in certain backwards countries with incredibly low-cost computer equipment. One inch is equal to 2.54 cm.

c
: Centimeters. One centimeter is equal to 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 Section 5.18.2 [Fractional Type Sizes], page 122, for a discussion of these units.

f
: Fractions. Value is 65536. See Section 5.28 [Colors], page 156, for usage.

The other measurements understood by gtroff depend on settings currently in effect in gtroff. These are very useful for specifying measurements which 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 groff, this is half of an em.

v
: Vertical space. This is equivalent to the current line spacing. See Section 5.18 [Sizes], page 119, for more information about this.

[1] those that specify vertical or horizontal motion or a type size

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 7i/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). See below for the use of unary operators in motion requests.
- 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 which 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 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.

'`+`' and '`-`' are also treated differently by the following 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 Section 5.6.1 [Setting Registers], page 70, for some examples.

`\B'`*anything*`'` [Escape]

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 Section 5.5.1.1 [Request and Macro Arguments], page 65, and Section 5.20 [Conditionals and Loops], page 127.

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 `groff` runs on a machine based on ASCII, causing a warning message of type '`input`' (see Section 5.33 [Debugging], page 167, for more details): `0x00`, `0x0B`, `0x0D-0x1F`, `0x80-0x9F`.

  And here are the invalid input characters if `groff` 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 which 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 which 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*`'` [Escape]

Test whether an identifier *ident* is valid in `gtroff`. 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 Section 5.5.3 [Escapes], page 67, for details on parameter delimiting characters.

Identifiers in `gtroff` can be any length, but, in some contexts, `gtroff` 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 (`gtroff` 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 `gtroff` finds an undefined identifier, it emits a warning, doing the following:

- If the identifier is a string, macro, or diversion, `gtroff` defines it as empty.
- If the identifier is a number register, `gtroff` defines it with a value of 0.

See Section 5.33.1 [Warnings], page 169., Section 5.6.2 [Interpolating Registers], page 71, and Section 5.19 [Strings], page 123.

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, `gtroff` 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 Section 5.6.2 [Interpolating Registers], page 71, and Section 5.19 [Strings], page 123.

5.5 Embedded Commands

Most documents need more functionality beyond filling, adjusting and implicit line breaking. In order to gain further functionality, `gtroff` allows commands to be embedded into the text, in two ways.

The first is a *request* which takes up an entire line, and does some large-scale operation (e.g. break lines, start new pages).

The other is an *escape* which 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 Section 5.11 [Character Translations], page 90, 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 Section 5.24.4 [Blank Line Traps], page 148.

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.

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.[2]

An argument to a macro which 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
```

[2] Plan 9's `troff` implementation also allows tabs for argument separation – `gtroff` intentionally doesn't support this.

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`'.[3]

A double quote which 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`):

  ```
  .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 `gtroff` preserves the input level.

- Use the double quote glyph `\(dq`. This works with and without compatibility mode enabled since `gtroff` doesn't convert `\(dq` back to a double quote input character.

  Not that this method won't work with UNIX `troff` in general since the glyph 'dq' isn't defined normally.

Double quotes in the `ds` request are handled differently. See Section 5.19 [Strings], page 123, for more details.

---

[3] 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 which can't stretch. `gtroff` provides a different command `\~` to insert a stretchable, non-breaking space.

5.5.2 Macros

`gtroff` has a *macro* facility for defining a series of lines which 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 Section 5.21 [Writing Macros], page 131, and Section 5.5.1.1 [Request and Macro Arguments], page 65.

5.5.3 Escapes

Escapes may occur anywhere in the input to `gtroff`. 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 Section 5.11 [Character Translations], page 90.

Escape sequences which 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 which 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
\o
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 Section 5.11 [Character Translations], page 90, for an exact description of those escapes.

See Section 5.34 [Implementation Differences], page 171, Section 5.21.1 [Copy-in Mode], page 134, and Section 5.25 [Diversions], page 149, Section 5.4 [Identifiers], page 62, for more information.

5.5.3.1 Comments

Probably one of the most[4] common forms of escapes is the comment.

`\"` [Escape]

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 idiosyncracy 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
```

[4] Unfortunately, this is a lie. But hopefully future `gtroff` hackers will believe it `:-)`

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.

`\#` [Escape]

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*] [Request]

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 Section 5.6.3 [Auto-increment], page 72).

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 Section 5.4 [Identifiers], page 62, for details on register identifiers.

5.6.1 Setting Registers

Define or set registers using the `nr` request or the `\R` escape.

`.nr` *ident value* [Request]
`\R'`*ident value*`'` [Escape]

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 Section 5.5.3 [Escapes], page 67, for details on parameter delimiting characters.

The `\R` escape doesn't produce an input token in `gtroff`; with 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
```

Both `nr` and `\R` have two additional special forms to increment or decrement a register.

`.nr` *ident* `+`*value* [Request]
`.nr` *ident* `-`*value* [Request]
`\R'`*ident* `+`*value*`'` [Escape]
`\R'`*ident* `-`*value*`'` [Escape]

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* [Request]

Remove number register *ident*. If *ident* doesn't exist, the request is ignored.

`.rnn` *ident1 ident2* [Request]

Rename number register *ident1* to *ident2*. If either *ident1* or *ident2* doesn't exist, the request is ignored.

`.aln` *ident1 ident2* [Request]

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 Section 5.33 [Debugging], page 167, for information about warnings.

5.6.2 Interpolating Registers

Numeric registers can be accessed via the `\n` escape.

`\n`*i* [Escape]
`\n(`*id* [Escape]
`\n[`*ident*`]` [Escape]

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* [Request]

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* [Escape]
`\n-`*i* [Escape]
`\n(+`*id* [Escape]
`\n(-`*id* [Escape]
`\n+(`*id* [Escape]
`\n-(`*id* [Escape]
`\n[+`*ident*`]` [Escape]
`\n[-`*ident*`]` [Escape]
`\n+[`*ident*`]` [Escape]
`\n-[`*ident*`]` [Escape]

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 in the text of an input file (as opposed to part of an expression), it is 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* [Request]

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 will do; `gtroff` only counts how many digits are specified. As a consequence, `af`'s default format '`1`' could be specified as '`0`' also (and exactly this is returned by the `\g` 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 Section 5.33 [Debugging], page 167, 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 '`I`' formats is 39999 (or −39999); UNIX `troff` uses '`z`' and '`w`' to represent 10000 and 5000 in Roman numerals, and so does `gtroff`. 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` [Escape]
`\g(id` [Escape]
`\g[ident]` [Escape]

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 which are not described elsewhere in this manual. Any register which begins with a '`.`' is read-only. A complete listing of all built-in registers can be found in tie E [Register Index], page 215.

`\n[.F]` This string-valued register returns the current input file name.

`\n[.H]` Horizontal resolution in basic units.

`\n[.U]` If `gtroff` is called with the '`-U`' command line option, the number register `.U` is set to 1, and zero otherwise. See Section 2.1 [Groff Options], page 7.

`\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 `gtroff`.

`\n[minutes]`
The number of minutes after the hour, in the range 0 to 59. Initialized at start-up of `gtroff`.

`\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.`' (a `gtroff` 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 Section 5.31 [Miscellaneous], page 162, 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 Section 2.1 [Groff Options], page 7.

`\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 Section 2.1 [Groff Options], page 7, 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 Section 2.1 [Groff Options], page 7.

`\*[.T]` A single read-write string register which 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 Section 5.1.5 [Implicit Line Breaks], page 58. 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` [Request]

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` [Request]
`\n[.u]` [Register]

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 Section 5.26 [Environments], page 153).

See Section 5.14 [Line Control], page 98, for interaction with the `\c` escape.

`.nf` [Request]

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 Section 5.26 [Environments], page 153).

See Section 5.14 [Line Control], page 98, for interaction with the `\c` escape.

`.ad` [*mode*] [Request]
`\n[.j]` [Register]

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.

With no argument, `gtroff` adjusts lines in the same way it did before adjusting was deactivated (with a call to `na`, for example).

```
text
.ad r
.nr ad \n[.j]
text
.ad c
text
.na
text
.ad         \" back to centering
text
.ad \n[ad]  \" back to right justifying
```

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 Section 5.26 [Environments], page 153).

`.na` [Request]

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 Section 5.26 [Environments], page 153).

`.brp` [Request]
`\p` [Escape]

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*] [Request]
`\n[.ss]` [Register]
`\n[.sss]` [Register]

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 Section 5.26 [Environments], page 153).

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 Section 5.34 [Implementation Differences], page 171).

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 which 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*] [Request]
`\n[.ce]` [Register]

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 Section 5.26 [Environments], page 153).

The following example demonstrates the differences. Here the input:

```
.ll 4i
.ce 1000
This is a small text fragment which shows the differences
between the `.ce' and the `.ad c' request.
.ce 0

.ad c
This is a small text fragment which shows the differences
between the `.ce' and the `.ad c' request.
```

And here the result:

```
  This is a small text fragment which
         shows the differences
between the `.ce' and the `.ad c' request.

  This is a small text fragment which
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 which 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*] [Request]
`\n[.rj]` [Register]

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 Section 5.26 [Environments], page 153).

5.8 Manipulating Hyphenation

Here a description of requests which influence hyphenation.

`.hy` [*mode*] [Request]
`\n[.hy]` [Register]

Enable hyphenation. The request has an optional numeric argument, *mode*, to restrict hyphenation if necessary:

`1` The default argument if *mode* is omitted. Hyphenate without restrictions. This is also the start-up value of `gtroff`.

`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.

Values in the previous table are additive. For example, the value 12 causes `gtroff` to neither hyphenate the last two nor the first two characters of a word.

The current hyphenation restrictions can be found in the read-only number register '`.hy`'.

The hyphenation mode is associated with the current environment (see Section 5.26 [Environments], page 153).

`.nh` [Request]

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 Section 5.26 [Environments], page 153).

`.hlm` [*nnn*] [Request]
`\n[.hlm]` [Register]
`\n[.hlc]` [Register]

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 Section 5.26 [Environments], page 153). 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* ... [Request]

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 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 `troff` there was a limited amount of space to store such information; fortunately, with `gtroff`, this is no longer a restriction.

`\%` [Escape]
`\:` [Escape]

To tell `gtroff` 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 `gtroff` 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*] [Request]

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 Section 5.26 [Environments], page 153).

`.hpf` *pattern_file* [Request]
`.hpfa` *pattern_file* [Request]
`.hpfcode` *a b* [*c d* ...] [Request]

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 '`-m`*name*' 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), `gtroff` won't hyphenate at all.

The `hpfa` request appends a file of patterns to the current list.

The `hpfcode` request defines mapping values for character codes in hyphenation patterns. `hpf` or `hpfa` 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 which would be invalid otherwise.

The set of hyphenation patterns is associated with the current language set by the `hla` request. The `hpf` request is usually invoked by the '`troffrc`' or '`troffrc-end`' file; by default, '`troffrc`' loads hyphenation patterns and exceptions for American English (in files '`hyphen.us`' and '`hyphenex.us`').

A second call to `hpf` (for the same language) will replace the hyphenation patterns with the new ones.

Invoking `hpf` causes an error if there is no current hyphenation language.

`.hcode` *c1 code1* [*c2 code2* ...] [Request]

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. With 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 which 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 which 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*] [Request]
`\n[.hym]` [Register]

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 Section 5.26 [Environments], page 153).

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*] [Request]
`\n[.hys]` [Register]

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 Section 5.26 [Environments], page 153).

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*] [Request]

Set the *soft hyphen character* to *glyph*.[5] If the argument is omitted, the soft hyphen character is set to the default glyph `\(hy` (this is the start-up value of `gtroff` 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.

---

[5] *Soft hyphen character* is a misnomer since it is an output glyph.

`.hla` *language* [Request]
`\n[.hla]` [Register]

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*] [Request]

Space downwards *distance*. With no argument it advances 1 line. A negative argument causes `gtroff` to move up the page the specified distance. If the argument is preceded by a '`|`' then `gtroff` moves that distance from the top of the page. This request causes a line break. The default scaling indicator is '`v`'.

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]` [Request]
`\n[.L]` [Register]

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 Section 5.26 [Environments], page 153).

The read-only number register `.L` contains the current line spacing setting.

See Section 5.18.1 [Changing Type Sizes], page 120, for the requests `vs` and `pvs` as alternatives to `ls`.

`\x'spacing'` [Escape]
`\n[.a]` [Register]

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 Section 5.5.3 [Escapes], page 67, 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` [Request]
`.rs` [Request]
`\n[.ns]` [Register]

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 Section 5.16 [Page Control], page 101, 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` [Escape]

This escape is a non-interpreted tab character. In copy mode (see Section 5.21.1 [Copy-in Mode], page 134), `\t` is the same as a real tab character.

`.ta` [*n1 n2* . . . *nn* `T` *r1 r2* . . . *rn*] [Request]
`\n[.tabs]` [Register]

Change tab stop positions. This request takes a series of tab specifiers as arguments (optionally divided into two groups with the letter '`T`') which 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 which 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 which 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 Section 5.26 [Environments], page 153).
- Calling `ta` without an argument removes all tab stops.
- The start-up value of `gtroff` is '`T 0.8i`'.

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*] [Request]

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 Section 5.26 [Environments], page 153).[6]

`.linetabs` *n* [Request]
`\n[.linetabs]` [Register]

If *n* is missing or not zero, enable *line-tabs* mode, or disable it otherwise (the default). In line-tabs mode, `gtroff` computes tab distances relative to the (current) output line instead of the input line.

For example, the following code:

[6] *Tab repetition character* is a misnomer since it is an output glyph.

```
.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 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 `gtroff` 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` [Escape]

This escape is a non-interpreted leader character. In copy mode (see Section 5.21.1 [Copy-in Mode], page 134), `\a` is the same as a real leader character.

`.lc` [*fill-glyph*] [Request]

Declare the *leader repetition character*.[7] Without an argument, leaders act the same as tabs (i.e., using whitespace for filling). `gtroff`'s start-up value is a dot (‘`.`’). The value of the leader repetition character is associated with the current environment (see Section 5.26 [Environments], page 153).

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]
```

[7] *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 which 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 TeX'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*]] [Request]

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 Section 5.26 [Environments], page 153).

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*] [Request]

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 Section 5.26 [Environments], page 153).

`.c2` [*c*] [Request]

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 Section 5.26 [Environments], page 153).

`.eo` [Request]

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*] [Request]

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 will likely break macro packages since `gtroff` has no mechanism to 'intern' macros, i.e., to convert a macro definition into an internal form which is independent of its representation (TeX has this mechanism). If a macro is called, it is executed literally.

`.ecs` [Request]
`.ecr` [Request]

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 `\`.

`\\` [Escape]
`\e` [Escape]
`\E` [Escape]

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'123'
..
.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'123'
..
.xxx
    ⇒ 1
```

`\.` [Escape]

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 exexuting 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 Section 5.32 [Gtroff Internals], page 165, for more on this process).

`.tr` *abcd...* [Request]
`.trin` *abcd...* [Request]

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 Section 5.25 [Diversions], page 149, 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 Section 5.32 [Gtroff Internals], page 165.
- `troff` no longer has a hard-coded dependency on Latin-1; all `char`*`XXX`* entities have been removed from the font description files. This has a notable consequence which 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 `gtroff` doesn't handle translations of translations.

  The right way to write the above translation is

  ```
  .tr é\['E]
  ```

  With 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...* [Request]

`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 Section 5.25 [Diversions], page 149, 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 GNU `troff`, 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 `gtroff`. 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 `gtroff` 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: `gtroff` provides two built-in conditions 'n' and 't' for the `if`, `ie`, and `while` requests to decide whether `gtroff` shall behave like `nroff` or like `troff`.

`.troff` [Request]

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 `gtroff` (*not* `groff`) is started with the '-R' switch to avoid loading of the start-up files 'troffrc' and 'troffrc-end'. Without '-R', `gtroff` stays in troff mode if the output device is not a TTY (e.g. 'ps').

`.nroff` [Request]

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 `gtroff` 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 Section 5.20 [Conditionals and Loops], page 127, for more details on built-in conditions.

5.13 Line Layout

The following drawing shows the dimensions which `gtroff` uses for placing a line of output onto the page. They are labeled with the request which manipulates each dimension.

```
              -->| in |<--
                 |<-----------ll------------>|
            +----+----+----------------------+----+
            |    :    :                      :    |
            +----+----+----------------------+----+
         -->| po |<--
            |<--------paper width---------------->|
```

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*] [Request]
`.po +`*offset* [Request]
`.po -`*offset* [Request]
`\n[.o]` [Register]

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*] [Request]
.in +*indent* [Request]
.in -*indent* [Request]
\n[.i] [Register]

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 Section 5.26 [Environments], page 153).

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* [Request]
.ti +*offset* [Request]
.ti -*offset* [Request]
\n[.in] [Register]

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 Section 5.26 [Environments], page 153). 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*] [Request]
`.ll +`*length* [Request]
`.ll -`*length* [Request]
`\n[.l]` [Register]
`\n[.ll]` [Register]

Set the line length to *length* (or increment or decrement the current value by *length*). Initially, the line length is set to 6.5 i. 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 Section 5.26 [Environments], page 153).

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 which act exactly at the place where called.

```
.\" A simple macro to underline a word
.de underline
.  nop \\$1\l'|0\[ul]'
..
```

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⟩` [Escape]
`\c` [Escape]
`\n[.int]` [Register]

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 will be 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` will be 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` will be 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 which 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 Section 5.26 [Environments], page 153).

5.15 Page Layout

`gtroff` provides some very primitive operations for controlling page layout.

`.pl` [*length*] [Request]
`.pl +`*length* [Request]
`.pl -`*length* [Request]
`\n[.p]` [Register]

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 Section 5.24 [Traps], page 144, for further information on how to do this.

Negative `pl` 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, `pl` sets the page length to 11 i.

`gtroff` provides several operations which help in setting up top and bottom titles (or headers and footers).

`.tl '`*left*`'`*center*`'`*right*`'` [Request]

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, `tl` is ignored.

Some notes:

- A title line is not restricted to the top or bottom of a page.
- `tl` prints the title line immediately, ignoring a partially filled line (which stays untouched).
- It is not an error to omit closing delimiters. For example, '`.tl /foo`' is equivalent to '`.tl /foo///`': It prints a title line with the left justified word '`foo`'; the centered and right justfied parts are empty.
- `tl` accepts the same parameter delimiting characters as the `\A` escape; see Section 5.5.3 [Escapes], page 67.

`.lt [`*length*`]` [Request]
`.lt +`*length* [Request]
`.lt -`*length* [Request]
`\n[.lt]` [Register]

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 6.5 i. 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 Section 5.26 [Environments], page 153).

`.pn` *page* [Request]
`.pn +`*page* [Request]
`.pn -`*page* [Request]
`\n[.pn]` [Register]

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*`]` [Request]

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 Section 5.24 [Traps], page 144.

5.16 Page Control

`.bp [`*page*`]` [Request]
`.bp +`*page* [Request]
`.bp -`*page* [Request]
`\n[%]` [Register]

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 Section 5.15 [Page Layout], page 100.

```
.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 Section 5.25 [Diversions], page 149).

The read-write register `%` holds the current page number.

The number register `.pe` is set to 1 while `bp` is active. See Section 5.24.1 [Page Location Traps], page 144.

`.ne` [*space*] [Request]

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 Section 5.24 [Traps], page 144, 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` will then automatically cause a page break if there is space for one line only.

`.sv` [*space*] [Request]
`.os` [Request]

`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` will ignore them.

`\n[nl]` [Register]

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 Section 5.25 [Diversions], page 149, 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 which contains various special symbols (Greek, mathematics).

5.17.1 Changing Fonts

`.ft` [*font*] [Request]
`\f`*f* [Escape]
`\f(`*fn* [Escape]
`\f[`*font*`]` [Escape]
`\n[.sty]` [Register]

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).

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 Section 5.17.3 [Font Positions], page 106, for an alternative syntax.

`.ftr` *f* [*g*] [Request]

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.

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`', '`-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 Section 2.1 [Groff Options], page 7).

`.fam` [*family*] [Request]
`\n[.fam]` [Register]

`\Ff` [Escape]
`\F(fm` [Escape]
`\F[family]` [Escape]
`\n[.fn]` [Register]

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` [Request]

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' will be 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 will instead be 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 Section 2.1 [Groff Options], page 7). 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 `gtroff` 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 Section 5.17.3 [Font Positions], page 106.

5.17.3 Font Positions

For the sake of old phototypesetters and compatibility with old versions of `troff`, `gtroff` has the concept of font *positions*, on which various fonts are mounted.

`.fp` *pos font* [*external-name*] [Request]
`\n[.f]` [Register]
`\n[.fp]` [Register]

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 `gtroff` 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 Section 5.26 [Environments], page 153).

```
.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 `gtroff` 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 `gtroff` 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* [Request]
`\f`*n* [Escape]
`\f(`*nn* [Escape]
`\f[`*nnn*`]` [Escape]

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 Section 5.17.1 [Changing Fonts], page 103, 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 which 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 `gtroff`, all glyph names of a particular font are defined in its font file. If the user requests a glyph not available in this font, `gtroff` 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 the exact rules how `gtroff` 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 `foo` at font position 3. We assume that `FOO` is a special font, containing glyph `foo`, 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 `foo` 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 foo which 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 gtroff 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 Section 8.2 [Font Files], page 192, and Section 5.17.5 [Special Fonts], page 114, 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 http://partners.adobe.com/asn/tech/type/unicodegn.jsp. The (frozen) set of glyph names which 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 will be named u*XXXX*[*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 will be named

  'u' *component1* '_' *component2* '_' *component3* ...

  Example: u0045_0302_0301.

  For simplicity, all Unicode characters which 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 which are composites itself. For example, u0100 (LATIN LETTER A WITH MACRON) will be 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` will be mapped to `^E`.

- glyph names of the GGL can't be used in composite glyph names; for example, `^E_u0301` is invalid.

`\(`*nm* [Escape]
`\[`*name*`]` [Escape]
`\[`*component1 component2 . . .*`]` [Escape]

Insert a symbol *name* (two-character name *nm*) or a composite glyph with component glyphs *component1*, *component2*, . . . There is no special syntax for one-character names – the natural form '`\`*n*' would collide with escapes.[8]

If *name* is undefined, a warning of type '`char`' is generated, and the escape is ignored. See Section 5.33 [Debugging], page 167, for information about warnings.

groff resolves `\[...]` with more than a single component as follows:

- Any component which is found in the GGL will be converted to the `u`*XXXX* form.
- Any component `u`*XXXX* which is found in the list of decomposable glyphs will be decomposed.
- The resulting elements are then concatenated with '`_`' inbetween, dropping the leading '`u`' in all elements but the first.

No check for the existence of any component (similar to `tr` request) will be done.

Examples:

`\[A ho]` '`A`' maps to `u0041`, '`ho`' maps to `u02DB`, thus the 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`.

[8] 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, `gtroff` has the special feature that `\[char`*XXX*`]` 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.

`\[^E u0301]`
`\[^E aa]`
`\[E a^ aa]`
`\[E ^ ']` ‘`^E`’ maps to `u0045_0302`, thus the final glyph name is `u0045_0302_0301` in all forms (assuming proper calls of the `composite` request).

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*`'` [Escape]

Typeset the glyph named *xxx*.[9] Normally it is more convenient to use `\[xxx]`, but `\C` has the advantage that it is compatible with newer versions of AT&T `troff` and is available in compatibility mode.

`.composite` *from to* [Request]

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*`'` [Escape]

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.

`\'` [Escape]

This is a backslash followed by the apostrophe character, ASCII character `0x27` (EBCDIC character `0x7D`). The same as `\[aa]`, the acute accent.

[9] `\C` is actually a misnomer since it accesses an output glyph.

`\‘` [Escape]

This is a backslash followed by ASCII character `0x60` (EBCDIC character `0x79` usually). The same as `\[ga]`, the grave accent.

`\-` [Escape]

This is the same as `\[-]`, the minus sign in the current font.

`.cflags` *n c1 c2 . . .* [Request]

Input characters and symbols have certain properties associated with it.[10] These properties can be modified with the `cflags` request. The first argument is the sum 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.

`1` The character ends sentences (initially characters ‘`.?!`’ have this property).

`2` Lines can be broken before the character (initially no characters have this property).

`4` Lines can be broken after the character (initially the character ‘`-`’ and the symbols ‘`\[hy]`’ and ‘`\[em]`’ have this property).

`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 TeX (initially characters ‘`"')]*`’ and the symbols ‘`\[dg]`’ and ‘`\[rq]`’ have this property).

`.char` *g* [*string*] [Request]
`.fchar` *g* [*string*] [Request]
`.fschar` *f g* [*string*] [Request]
`.schar` *g* [*string*] [Request]

Define a new glyph *g* to be *string* (which can be empty).[11] 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

[10] Note that the output glyphs themselves don’t have such properties. For `gtroff`, 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.

[11] `char` is a misnomer since an output glyph is defined.

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: `gtroff` only checks for glyphs defined with `fchar` if it cannot find the glyph in the current font. `gtroff` carries out this test before checking special fonts.

`fschar` defines a fallback glyph for font *f*: `gtroff` 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.

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 Section 5.17.4 [Using Symbols], page 108, for a detailed description of the glyph searching mechanism in `gtroff`.

`.rchar` *c1 c2 . . .* [Request]
`.rfschar` *f c1 c2 . . .* [Request]

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 Section 7.1 [Special Characters], page 177.

5.17.5 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 Section 5.17.4 [Using Symbols], page 108, for a detailed description of the glyph searching mechanism in `gtroff`.

Usually, only non-TTY devices have special fonts.

`.special` [*s1 s2 . . .*] [Request]
`.fspecial` *f* [*s1 s2 . . .*] [Request]

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 which appear in a call to `special` or `fspecial` are loaded.

See Section 5.17.4 [Using Symbols], page 108, for the exact search order of glyphs.

5.17.6 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*`'` [Escape]
`\H'+`*height*`'` [Escape]
`\H'-`*height*`'` [Escape]
`\n[.height]` [Register]

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`’ device supports 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*`'` [Escape]
`\n[.slant]` [Register]

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`' device supports this feature.

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'
```

This request is incorrectly documented in the original UNIX troff manual; the slant is always set to an absolute value.

`.ul` [*lines*] [Request]

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 Section 5.26 [Environments], page 153). The underline font can be changed with the `uf` request.

The `ul` request does not underline spaces.

`.cu` [*lines*] [Request]

The `cu` request is similar to `ul` but underlines spaces as well (if a TTY output device is used).

`.uf` *font* [Request]

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*] [Request]
`.bd` *font1 font2* [*offset*] [Request]
`\n[.b]` [Register]

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*]] [Request]

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.

5.17.7 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 `troff` also supported 'ff', 'ffi', and 'ffl' ligatures. Advanced typesetters or 'expert' fonts may include ligatures for 'ft' and 'ct', although GNU `troff` 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*] [Request]
`\n[.lg]` [Register]

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*] [Request]
`\n[.kern]` [Register]

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: '`V\&A`'.

See Section 8.2.2 [Font File Format], page 194.

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* [Request]

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 are needed. There are two escapes to help with this.

`\/` [Escape]

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 )
```

`\,` [Escape]

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
    ⇒ q f
```

`\&` [Escape]

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 Section 5.11 [Character Translations], page 90).

`\)` [Escape]

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

`gtroff` uses two dimensions with each line of text, type size and vertical spacing. The *type size* is approximately the height of the tallest glyph.[12] *Vertical spacing* is the amount of space `gtroff` 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, `gtroff` 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').

---

[12] 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 10 pt; 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.

5.18.1 Changing Type Sizes

`.ps` [*size*] [Request]
`.ps +`*size* [Request]
`.ps -`*size* [Request]
`\s`*size* [Escape]
`\n[.s]` [Register]

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, or no argument, goes back to the previous size.

Default scaling indicator of `size` is '`z`'. If `size` is zero or 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 Section 5.26 [Environments], page 153).

```
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:

`\s`*n* Set the point size to *n* points. *n* must be either 0 or in the range 4 to 39.

`\s+`*n*
`\s-`*n* Increase or decrease the point size by *n* points. *n* must be exactly one digit.

`\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 Section 5.18.2 [Fractional Type Sizes], page 122, for yet another syntactical form of using the `\s` escape.

`.sizes` *s1 s2 . . . sn* [*0*] [Request]

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*] [Request]
`.vs +`*space* [Request]
`.vs -`*space* [Request]
`\n[.v]` [Register]

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 Section 5.26 [Environments], page 153).

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 which 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 Section 5.9 [Manipulating Spacing], page 84, for more details on the `\x` escape and the `ls` request.

`.pvs` [*space*] [Request]
`.pvs +`*space* [Request]
`.pvs -`*space* [Request]
`\n[.pvs]` [Register]

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 Section 5.26 [Environments], page 153).

5.18.2 Fractional Type Sizes

A *scaled point* is equal to 1/*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]` [Register]

A read-only number register returning the point size in scaled points.

`.ps` is associated with the current environment (see Section 5.26 [Environments], page 153).

`\n[.psr]` [Register]
`\n[.sr]` [Register]

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 10.95 pt instead, this value is actually used.

Both registers are associated with the current environment (see Section 5.26 [Environments], page 153).

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 or decrease the point size by *n* scaled points; *n* is a numeric expression with a default scale indicator of '`z`'.

See Section 8.2 [Font Files], page 192.

5.19 Strings

`gtroff` has string variables, which are entirely for user convenience (i.e. there are no built-in strings exept `.T`, but even this is a read-write string variable).

`.ds` *name* [*string*] [Request]
`.ds1` *name* [*string*] [Request]
`\*`*n* [Escape]
`\*(`*nm* [Escape]
`\*[`*name arg1 arg2 ...*`]` [Escape]

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 which are handled identically to macro arguments; the single exception is that a closing bracket as an argument must be enclosed in double quotes. See Section 5.5.1.1 [Request and Macro Arguments], page 65, and Section 5.21.2 [Parameters], page 135.

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 `gtroff`. 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 Section 5.33 [Debugging], page 167, for more details).

Caution: Unlike other requests, the second argument to the `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 ix \\n[xxx].
.
.cp 1
.
\*(aa
    ⇒ warning: number register '[' not defined
    ⇒ The value of xxx is 0xxx].
\*(bb
    ⇒ The value of xxx ix 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.
```

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 UNIX `troff`, 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 Section 5.32 [Gtroff Internals], page 165, for more information.

`.as` *name* [*string*] [Request]
`.as1` *name* [*string*] [Request]

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*] [Request]

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 taken to be equal to the string's length. 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
```

`.length` *reg str* [Request]

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* [Request]

Rename the request, macro, diversion, or string *xx* to *yy*.

`.rm` *xx* [Request]

Remove the request, macro, diversion, or string *xx*. `gtroff` treats subsequent invocations as if the object had never been defined.

`.als` *new old* [Request]

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, `gtroff` generates a warning of type '`mac`' and ignores the request.

`.chop` *xx* [Request]

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 Section 5.32 [Gtroff Internals], page 165, for details on nodes inserted additionally by `gtroff`.

See Section 5.4 [Identifiers], page 62, and Section 5.5.3.1 [Comments], page 68.

5.20 Conditionals and Loops

5.20.1 Operators in Conditionals

In `if` and `while` requests, 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).

`t` True if the document is being processed in troff mode (i.e., the `.troff` command has been issued).

`v` Always false. This condition is for compatibility with other `troff` versions only (identifying a `-Tversatec` device).

`'`*xxx*`'`*yyy*`'`

True if the string *xxx* is equal to the string *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 Section 5.5.3 [Escapes], page 67). `gtroff` formats the strings before being compared:

```
.ie "|"\fR|\fP" \
true
.el \
false
    ⇒ true
```

The resulting motions, glyph sizes, and fonts have to match,[13] 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 I`' had been added before the '`.ie`', the result would be "false" because (the first) '`|`' produces an italic '`|`' rather than a roman one.

`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*.

`m` *xxx* True if there is a color named *xxx*.

`c` *g* True if there is a glyph *g* available[14]; *g* is either an ASCII character or a special character (`\(`*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 (this 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.

---

[13] The created output nodes must be identical. See Section 5.32 [Gtroff Internals], page 165.

[14] The name of this conditional operator is a misnomer since it tests names of output glyphs.

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 Section 5.3 [Expressions], page 61.

5.20.2 if-else

`gtroff` has if-then-else constructs like other languages, although the formatting can be painful.

`.if` *expr anything* [Request]

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 Section 5.3 [Expressions], page 61, for more info.

```
.nr xxx 1
.nr yyy 2
.if ((\n[xxx] == 1) & (\n[yyy] == 2)) true
    ⇒ true
```

`.nop` *anything* [Request]

Executes *anything*. This is similar to `.if 1`.

`.ie` *expr anything* [Request]
`.el` *anything* [Request]

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
```

`\{` [Escape]

`\}` [Escape]

In many cases, an if (or if-else) construct needs to execute more than one request. This can be done using the `\{` and `\}` escapes. The following example shows the possible ways to use these escapes (note the position of the opening and closing braces).

```
.ie t \{\
.    ds lq ``
.    ds rq ''
.\}
.el \
.\{\
.    ds lq "
.    ds rq "\}
```

See Section 5.3 [Expressions], page 61.

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* [Request]

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 which 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 ofter better solution (UNIX `troff` doesn't have the `while` request) is to use a recursive macro instead which 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` [Request]
Break out of a `while` loop. Be sure not to confuse this with the `br` request (causing a line break).

`.continue` [Request]
Finish the current iteration of a `while` loop, immediately restarting the next iteration.

See Section 5.3 [Expressions], page 61.

5.21 Writing Macros

A *macro* is a collection of text and embedded commands which can be invoked multiple times. Use macros to define common operations.

`.de` *name* [*end*] [Request]
`.de1` *name* [*end*] [Request]
`.dei` *name* [*end*] [Request]
`.dei1` *name* [*end*] [Request]
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). The optional second argument to `de` changes this to a macro to '`.`*`end`*'.

There can be whitespace after the first dot in the line containing the ending token (either '`.`' or macro '*`end`*').

Here a small example macro called '`P`' which 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]Hallo \\\\$1!\f[]
.  end
..
.
.foo
.bar Joe
```

⇒ **Hallo Joe!**

Since `\f` has no expansion, it isn't necessary to protect its backslash. Had we defined another macro within `bar` which 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 ix \\n[xxx].
..
.
.cp 1
.
.aa
    ⇒ warning: number register `[' not defined
    ⇒ The value of xxx is 0xxx].
.bb
    ⇒ The value of xxx ix 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 Section 5.32 [Gtroff Internals], page 165, 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.

.am *name* [*end*] [Request]
.am1 *name* [*end*] [Request]
.ami *name* [*end*] [Request]
.ami1 *name* [*end*] [Request]

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 Section 5.19 [Strings], page 123, for the `als` request to rename a macro.

The `de`, `am`, `di`, `da`, `ds`, and `as` requests (together with its variants) only create a new object if the name of the macro, diversion or string diversion is currently undefined or if it is defined to be a request; normally they modify the value of an existing object.

`.return` [*anything*] [Request]

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 `gtroff` 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[.$]` [Register]

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* [Escape]
`\$(`*nn* [Escape]
`\$[`*nnn*`]` [Escape]

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*] [Request]

Shift the arguments 1 position, or as many positions as specified by its argument. After executing this request, argument i becomes argument $i - n$; arguments 1 to n are no longer available. Shifting by negative amounts is currently undefined.

The register `.$` is adjusted accordingly.

`\$*` [Escape]
`\$@` [Escape]

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 Section 5.5.1.1 [Request and Macro Arguments], page 65).

`\$0` [Escape]

The name used to invoke the current macro. The `als` request can make a macro have more than one name.

```
.de generic-macro
.  ...
.  if \\n[error] \{\
.    tm \\$0: Houston, we have a problem.
.    return
.  \}
..
.
.als foo generic-macro
.als bar generic-macro
```

See Section 5.5.1.1 [Request and Macro Arguments], page 65.

5.22 Page Motions

See Section 5.9 [Manipulating Spacing], page 84, for a discussion of the main request for vertical motion, `sp`.

`.mk` [*reg*] [Request]
`.rt` [*dist*] [Request]

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 which 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 which 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*`'` [Escape]

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 which 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` [Escape]
Move upwards 1 v.

`\u` [Escape]
Move upwards .5 v.

`\d` [Escape]
Move down .5 v.

`\h'`*e*`'` [Escape]
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⟩ [Escape]
An unbreakable and unpaddable (i.e. not expanded during filling) space. (Note: This is a backslash followed by a space.)

`\~` [Escape]
An unbreakable space that stretches like a normal inter-word space when a line is adjusted.

`\|` [Escape]
A 1/6 th em space. Ignored for TTY output devices (rounded to zero).

`\^` [Escape]
A 1/12 th em space. Ignored for TTY output devices (rounded to zero).

`\0` [Escape]
A space the size of a digit.

The following string sets the TeX logo:

```
.ds TeX T\h'-.1667m'\v'.224m'E\v'-.224m'\h'-.125m'X
```

`\w'`*text*`'` [Escape]
`\n[st]` [Register]
`\n[sb]` [Register]
`\n[rst]` [Register]
`\n[rsb]` [Register]

`\n[ct]` [Register]
`\n[ssc]` [Register]
`\n[skw]` [Register]

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:

`st`
`sb` The highest and lowest point of the baseline, respectively, in *text*.

`rst`
`rsb` Like the `st` and `sb` registers, but takes account of the heights and depths of glyphs. With other words, this gives the highest and lowest point of *text*. Values below the baseline are negative.

`ct` Defines the kinds of glyphs occurring in *text*:

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.

`ssc` The amount of horizontal space (possibly negative) that should be added to the last glyph before a subscript.

`skw` How far to right of the center of the last glyph in the `\w` argument, the center of an accent from a roman font should be placed over that glyph.

`\k`*p* [Escape]
`\k(`*ps* [Escape]
`\k[`*position*`]` [Escape]

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]` [Register]

The current horizontal position at the input line.

`\n[.k]` [Register]

A read-only number register containing the current horizontal output position (relative to the current indentation).

`\o'abc'` [Escape]

Overstrike glyphs *a*, *b*, *c*, ...; the glyphs are centered, and the resulting spacing is the largest width of the affected glyphs.

`\zg` [Escape]

Print glyph *g* with zero width, i.e., without spacing. Use this to overstrike glyphs left-aligned.

`\Z'anything'` [Escape]

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 Section 5.22 [Page Motions], page 136, 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 Section 6.3 [gpic], page 175, and Section 6.4 [ggrn], page 175, for more information.

All drawing is done via escapes.

`\l'l'` [Escape]
`\l'lg'` [Escape]

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'|0\[rn]'\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*`'` [Escape]
`\L'`*lg*`'` [Escape]

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 the result, produced with `grotty`.

```
This is a
          |
          |
          |test.
```

`\D'`*command arg* . . .`'` [Escape]

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 `defcolor` request.

`\D'l` *dx dy*`'`

Draw a line from the current location to the relative point specified by (dx,dy), where positive values mean down and right, 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 will be always rounded to the next integer multiple of the horizontal resolution (the value of the `hor` keyword in the '`DESC`' file). Use `\M` (see Section 5.28 [Colors], page 156) 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'
..
```

`\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 `defcolor` request (see Section 5.28 [Colors], page 156); 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 Section 8.1.2.3 [Graphics Commands], page 184.

`\b'`*string*`'` [Escape]

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 unflexible positioning algorithm doesn't work with '`-Tdvi`' since the bracket pieces vary in height for this device. Instead, use the `eqn` preprocessor.

See Section 5.9 [Manipulating Spacing], page 84, how to adjust the vertical spacing with the `\x` escape.

5.24 Traps

Traps are locations, which, 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 `gtroff` 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* [Request]
`\n[.vpt]` [Register]

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*] [Request]

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`'.

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.

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]` [Register]

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 which can be represented in `groff`) if there are no diversion traps.

`.ch` *macro* [*dist*] [Request]

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]` [Register]

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 Section 5.16 [Page Control], page 101, 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]` [Register]

A read-only register containing the amount of vertical space truncated 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]` [Register]

A read-only register which 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) which tries to change the font on the current page, the effect will not be 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*] [Request]

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 Section 5.25 [Diversions], page 149, for more information.

5.24.3 Input Line Traps

`.it` *n macro* [Request]
`.itc` *n macro* [Request]

Set an input line trap. *n* is the number of lines of input which 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 which 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 Section 5.26 [Environments], page 153); switching to another environment disables the current input trap, and going back reactivates it, restoring the number of already processed lines.

5.24.4 Blank Line Traps

`.blm` *macro* [Request]

Set a blank line trap. `gtroff` executes *macro* when it encounters a blank line in the input file.

5.24.5 End-of-input Traps

`.em` *macro* [Request]

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
.  ne 5v
.  sp |(\\n[.t] - 6v)
.  in +4i
.  lc _
.  br
Approved:\t\a
.  sp
Date:\t\t\a
..
.
.em approval
```

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).

`.di` *macro* [Request]
`.da` *macro* [Request]

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.

`.box` *macro* [Request]
`.boxa` *macro* [Request]

Begin (or appends 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]` [Register]
`\n[.d]` [Register]

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]` [Register]

The *high-water mark* on the current page. 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]` [Register]
`\n[dl]` [Register]

After completing a diversion, the read-write number registers `dn` and `dl` contain the vertical and horizontal size of the diversion. Note that only the just processed lines are counted: For the computation of `dn` and `dl`, the requests `da` and `boxa` are handled as if `di` and `box` had been used – lines which 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
```

`\!` [Escape]
`\?`*anything*`\?` [Escape]

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 which 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 which 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* [Request]

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 which does something with the output before sending it to the output device, filtering out *string* again.

`.asciify` *div* [Request]

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
```

See Section 5.21.1 [Copy-in Mode], page 134.

`.unformat` *div* [Request]

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 Section 5.4 [Identifiers], page 62, for more info). Old versions of `troff` only had environments named '`0`', '`1`', and '`2`'.

`.ev` [*env*] [Request]
`\n[.ev]` [Register]

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* [Request]

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]` [Register]
`\n[.cht]` [Register]
`\n[.cdp]` [Register]
`\n[.csk]` [Register]

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] [Register]

The \n[.n] register contains the length of the previous output line in the current environment.

5.27 Suppressing output

\O*num* [Escape]

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 tie E [Register Index], page 215. These four registers mark the top left and bottom right hand corners of a box which 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.

'\O4' End a nesting level.

'\O[5*Pfilename*]'
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* will be associated with the production of the next inline image.

5.28 Colors

`.color` [*n*] [Request]
`\n[.color]` [Register]

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* [Request]

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), `##ffff0000ffff` (magenta). The default color name 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*] [Request]
`\m`*c* [Escape]
`\m(`*co* [Escape]
`\m[`*color*`]` [Escape]
`\n[.m]` [Register]

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 Section 5.26 [Environments], page 153).

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*] [Request]
`\M`*c* [Escape]
`\M(`*co* [Escape]
`\M[`*color*`]` [Escape]
`\n[.M]` [Register]

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 Section 5.26 [Environments], page 153).

Note that `\M` doesn't produce an input token in `gtroff`.

5.29 I/O

`gtroff` has several requests for including files:

`.so` *file* [Request]

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 Section 6.7 [gsoelim], page 175, 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* [Request]

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 `groff`'s or `troff`'s '`-U`' option to activate unsafe mode.

The comment regarding a final newline for the `so` request is valid for `pso` also.

`.mso` *file* [Request]

Identical to the `so` request except that `gtroff` searches for the specified *file* in the same directories as macro files for the 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.

`.trf` *file* [Request]
`.cf` *file* [Request]

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

```
.di x
.trf f
.di
```

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 `troff`, 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 `gtroff` input characters (see Section 5.4 [Identifiers], page 62).

Both requests cause a line break.

`.nx` [*file*] [Request]

Force `gtroff` 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* ...]] [Request]

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* [Request]

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* [Request]
`\n[systat]` [Register]

Execute the shell command(s) specified by *cmds*. The output is not saved anyplace, 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 which 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* [Request]
`.opena` *stream file* [Request]

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* [Request]

`.writec` *stream data* [Request]

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* [Request]

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* [Request]

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
```

`\V`*e* [Escape]
`\V(`*ev* [Escape]
`\V[`*env*`]` [Escape]

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 which give information directly to the postprocessor. This is particularly useful for embedding PostScript into the final document.

`\X'`*xxx*`'` [Escape]

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.

If the ‘use_charnames_in_special’ keyword is set in the ‘DESC’ file, special characters no longer cause an error; the name *xx* is represented as ‘\(xx)’ in the ‘x X’ output command. Additionally, the backslash is represented as \\.

‘use_charnames_in_special’ is currently used by grohtml only.

\Y*n* [Escape]
\Y(*nm* [Escape]
\Y[*name*] [Escape]

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 troff output format, and confuses drivers that do not know about this extension (see Section 8.1.2.4 [Device Control Commands], page 187).

See Chapter 7 [Output Devices], page 177.

5.31 Miscellaneous

This section documents parts of gtroff which cannot (yet) be categorized elsewhere in this manual.

.nm [*start* [*inc* [*space* [*indent*]]]] [Request]

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 which 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 which 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 Section 5.26 [Environments], page 153). 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*] [Request]

Temporarily turn off line numbering. The argument is the number of lines not to be numbered; this defaults to 1.

`.mc` *glyph* [*dist*] [Request]

Print a *margin character* to the right of the text.[15] 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

[15] *Margin character* is a misnomer since it is an output glyph.

```
.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 Section 5.26 [Environments], page 153).

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
\&
.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* [Request]
`\n[llx]` [Register]
`\n[lly]` [Register]
`\n[urx]` [Register]
`\n[ury]` [Register]

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*.[16] 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 Section 5.4 [Identifiers], page 62, 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 which have been passed unprocessed, and a list of output nodes. Consider the following the diversion.

```
.di xxx
a
\!b
c
.br
.di
```

It contains these elements.

node list	token list	element number
line start node	—	1
glyph node `a`	—	2
word space node	—	3
—	`b`	4
—	`\n`	5
glyph node `c`	—	6

[16] Except the escapes `\f`, `\F`, `\H`, `\m`, `\M`, `\R`, `\s`, and `\S` which are processed immediately if not in copy-in mode.

vertical size node	—	7
vertical size node	—	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 which 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`' which 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*] [Request]

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 which 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* [Request]
.tm1 *string* [Request]
.tmc *string* [Request]

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*] [Request]

Similar to the tm request, except that it causes gtroff to stop processing. With no argument it prints 'User Abort.' to standard error.

.ex [Request]

The ex request also causes gtroff to stop processing; see also Section 5.29 [I/O], page 157.

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 will be many errors and no useful output, gtroff can be forced to suppress formatted output with the '-z' flag.

`.pm` [Request]

Print the entire symbol table on `stderr`. Names of all defined macros, strings, and diversions are print 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` [Request]

Print the names and contents of all currently defined number registers on `stderr`.

`.ptr` [Request]

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` [Request]

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` [Request]

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]` [Register]

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* [Request]

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*] [Request]

Make `gtroff` 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
```

will cause a warning if `gtroff` 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`’).

`gtroff` 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*] [Request]
`\n[.warn]` [Register]

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 `gtroff` 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.[17] This is enabled by default.

[17] `char` is a misnomer since it reports missing glyphs – there aren’t missing input characters, only invalid ones.

'number'
'2' Invalid numeric expressions. This is enabled by default. See Section 5.3 [Expressions], page 61.

'break'
'4' In fill mode, lines which 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.

'el'
'16' Use of the `el` request with no matching `ie` request. See Section 5.20.2 [if-else], page 129.

'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.

'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 which 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*] [Request]
`.do` *cmd* [Request]
`\n[.C]` [Register]

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.

```
.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
Hallo!
..
\fB.xx\fP
```

prints '`Hallo!`' 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 Section 5.4 [Identifiers], page 62) 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 Section 5.18.2 [Fractional Type Sizes], page 122, 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`.[18] To store, for some reason, an escape sequence in a diversion that will be interpreted when the diversion is reread, either use the traditional `\!` transparent output facility, or, if this is unsuitable, the new `\?` escape sequence.

See Section 5.25 [Diversions], page 149, and Section 5.32 [Gtroff Internals], page 165, for more information.

[18] 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 `geqn`

6.1.1 Invoking `geqn`

6.2 `gtbl`

6.2.1 Invoking `gtbl`

6.3 `gpic`

6.3.1 Invoking `gpic`

6.4 `ggrn`

6.4.1 Invoking `ggrn`

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 `grefer`

6.6.1 Invoking `grefer`

6.7 `gsoelim`

6.7.1 Invoking `gsoelim`

7 Output Devices

7.1 Special Characters

See Section 8.2 [Font Files], page 192.

7.2 grotty

7.2.1 Invoking grotty

7.3 grops

7.3.1 Invoking grops

7.3.2 Embedding PostScript

7.4 grodvi

7.4.1 Invoking grodvi

7.5 grolj4

7.5.1 Invoking grolj4

7.6 grolbp

7.6.1 Invoking grolbp

7.7 grohtml

7.7.1 Invoking grohtml

7.7.2 grohtml specific registers and strings

`\n[ps4html]` [Register]
`\*[www-image-template]` [String]

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
```

The PostScript device is used to create all the image files, 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.

7.8 `gxditview`

7.8.1 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.[1]

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

[1] The parser and postprocessor for intermediate output can be found in the file '*groff-source-dir*`/src/libs/libdriver/input.cpp`'.

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 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 a *syntactical 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 to stack 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 splitted 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 will always be in the current font.

A string argument is always terminated by the next whitespace character (space, tab, or newline); an embedded '`#`' character is regarded as part of the argument, not as the beginning of a comment command. An integer argument is already terminated by the next non-digit character, which then is regarded as the first character of the next argument or command.

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 Section 8.1.2.4 [Device Control Commands], page 187. 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;[2] 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.

`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 analoguous 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*.

---

[2] '`c`' is actually a misnomer since it outputs a glyph.

`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 Section 8.1.4 [Output Language Compatibility], page 191.

`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 Section 8.2.1 [DESC File Format], page 192).

`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 Section 8.2.1 [DESC File Format], page 192).

`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 Section 5.23 [Drawing Requests], page 140.

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 analoguous 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.

$0 \le n \le 1000$
: Set the color for filling solid drawing objects to a shade of gray, where 0 corresponds to solid white, 1000 (the default) to solid black, and values in between to intermediate shades of gray; this is obsoleted by command '`DFg`'.

$n < 0$ or $n > 1000$
: Set the filling color to the color that is currently being used for the text and the outline, see command '`m`'. For example, the command sequence

```
mg 0 0 65536
Df -1
```

sets all colors to blue.

No position changing. This command is a `gtroff` extension.

`Dl` *`h v`*⟨line break⟩

Draw line from current position to offset (h,v) (integers in basic units '`u`'); then set current position to the end of the drawn line.

`Dp` *`h1 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. This command is a `gtroff` extension.

`Dp` *h1 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. 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. 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 newline 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 Section 8.1.1.1 [Separation], page 179).

`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 Section 5.17.3 [Font Positions], page 106.

`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 Section 8.1.4 [Output Language Compatibility], page 191.

`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 *gtroff*, 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 `gtroff` 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 `gtroff` escape sequence `\X`. The line-continuing feature is a `gtroff` 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.

ddg 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.

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
ch07e07l03lw06w11o07r05l03dh7
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-indenpendent 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.[3] The font files for device *name* are stored in a directory '`dev`*`name`*'. 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 comes last (if at all), the order of the lines is not important.

`res` *`n`*
: There are *n* machine units per inch.

`hor` *`n`*
: The horizontal resolution is *n* machine units. All horizontal quantities are rounded to be multiples of this value.

`vert` *`n`*
: The vertical resolution is *n* machine units. All vertical quantities are rounded to be multiples of this value.

`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 Section 5.18.2 [Fractional Type Sizes], page 122, for more information.

[3] Plan 9 `troff` has also abandoned the binary format.

`unitwidth` *n*
: Quantities in the font files are given in machine units for fonts whose point size is *n* scaled points.

`prepro` *program*
: Call *program* as a preprocessor. Currently, this keyword is used by `groff` with option '`-Thtml`' only.

`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).

`tcommand`
: This means that the postprocessor can handle the '`t`' and '`u`' intermediate output commands.

`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.

`styles` *S1 S2* ... *Sm*
: The first *m* font positions are associated with styles *S1* ... *Sm*.

`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.

`family` *fam*
: The default font family is *fam*.

`use_charnames_in_special`
: This command indicates that `gtroff` should encode special characters inside special commands. Currently, this is only used by the HTML output device. See Section 5.30 [Postprocessor Access], page 161.

`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 which 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.

`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 HTML output device.

`print` *program*
: Use *program* as a spooler program for printing. If omitted, the '`-l`' and '`-L`' options of `groff` are ignored.

`charset`
: This line and everything following in the file are ignored. It is allowed for the sake of backwards compatibility.

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 which 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* ... *lign* [0]
: Glyphs *lig1*, *lig2*, ..., *lign* 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.[4] 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[5]: 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 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 '`char`*n*' 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 '`char`*n*' 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, '`(`'.

The *code* field gives the code which the postprocessor uses to print the glyph. The glyph can also be input to `gtroff` using this code by means

---

[4] This keyword is misnamed since it starts a list of ordered glyphs, not characters.

[5] 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.

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 which the postprocessor uses to print the `gtroff` glyph *name*. This field is optional and has been introduced so that the HTML device driver can encode its character set. For example, the glyph '`\[Po]`' is represented as '`£`' in HTML 4.0.

Anything on the line after the *entity-name* field resp. after '`--`' will be 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

A Copying This Manual

A.1 GNU Free Documentation License

Version 1.2, November 2002

0. 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. Secondarily, 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.

1. 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.

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.

2. VERBATIM COPYING

   You may copy and distribute the Document in any medium, either commercially or noncommercially, 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.

3. 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.

4. 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:

   A. 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.

   B. 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.

   C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

   D. Preserve all the copyright notices of the Document.

   E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

   F. 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.

   G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

   H. Include an unaltered copy of this License.

   I. 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.

J. 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.

K. 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.

L. 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.

M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

O. 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.

5. 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."

6. 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.

7. 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.

8. 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.

9. TERMINATION

   You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. 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.

A.1.1 ADDENDUM: How to use this License for your documents

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 1.2
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.

B Request Index

Requests appear without the leading control character (normally either '.' or ''').

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

R

S

C Escape Index

Any escape sequence `\X` with *X* not in the list below emits a warning, printing glyph *X*.

D Operator Index

E Register Index

The macro package or program a specific register belongs to is appended in brackets.

A register name `x` consisting of exactly one character can be accessed as '`\nx`'. A register name `xx` consisting of exactly two characters can be accessed as '`\n(xx`'. Register names `xxx` of any length can be accessed as '`\n[xxx]`'.

Y

F Macro Index

The macro package a specific macro belongs to is appended in brackets. They appear without the leading control character (normally '.').

[

]

1

2

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

X

G String Index

The macro package or program a specific string belongs to is appended in brackets.

A string name x consisting of exactly one character can be accessed as '*x'. A string name xx consisting of exactly two characters can be accessed as '*(xx'. String names xxx of any length can be accessed as '*[xxx]'.

H

L

M

O

Q

R

S

T

U

V

W

H Glyph Name Index

A glyph name `xx` consisting of exactly two characters can be accessed as '`\(xx`'. Glyph names `xxx` of any length can be accessed as '`\[xxx]`'.

I Font File Keyword Index

J Program and File Index

K Concept Index

|

8

A

D

E

F

G

H

I

J

K

L

M

N

O

P

T

U

V

W

Y

Z