groff

The GNU implementation of `troff`
Edition 1.23.0
Autumn 2020

by Trent A. Fisher
and Werner Lemberg

This manual documents GNU `troff` version 1.23.0.

Table of Contents

1 Introduction

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

1.1 What Is `groff`?

`groff` belongs to an older generation of document preparation systems, which operate more like compilers than the more recent interactive 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.

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

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

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

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

1.2 History

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

[1] What You See Is What You Get

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

(1969), Doug McIlroy rewrote an extended and simplified version of `roff` in the BCPL programming language.

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

When they needed a more flexible language, a new version of `roff` called `nroff` (after "new `roff`", pronounced "en-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 "tee-roff" rather than "trough".

With `troff` came `nroff` (by 1974, they were actually the same program except for some '`#ifdef`'s), which was for producing output for line printers and character terminals. It understood everything `troff` did, and ignored the commands that were not applicable (e.g., font changes).

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

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

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

Brian Kernighan took on the task of rewriting `troff`. The result produced device-independent code that was easy for postprocessors to read and translate to appropriate printer commands. This new "device-independent `troff`", called `ditroff` by some, had several extensions, including drawing commands for lines, circles, ellipses, arcs, and B-splines[3].

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

[3] Short for "basis splines"; ask your local numerical analyst. The rest of us can just think of them as "curves".

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 device-independent `troff` in early 1989. The first version, `groff` 0.3.1, was released June 1990. `groff` included:

- A replacement for AT&T device-independent `troff` with many extensions.
- The `soelim`, `pic`, `tbl`, and `eqn` preprocessors.
- Postprocessors for character devices, PostScript, TEX's device-independent format (DVI), and the X Window System (X11). GNU `troff` also eliminated the need for a separate `nroff` program with a postprocessor to produce output for ASCII terminals.
- A version of the `me` macros and an implementation of the `man` macros.

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

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

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

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

1.3 `groff` Capabilities

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

- text filling, adjusting, and centering
- hyphenation
- page control
- font and glyph size control
- vertical spacing (e.g. double-spacing)
- line length and indenting
- macros, strings, diversions, and traps
- number registers
- tabs, leaders, and fields
- input and output conventions and character translation

- overstrike, bracket, line drawing, and zero-width functions
- local horizontal and vertical motions and the width function
- three-part titles
- output line numbering
- conditional acceptance of input
- environment switching
- insertions from the standard input
- input/output file switching
- output and error messages

1.4 Macro Packages

Since `groff` provides such low-level facilities, it can be quite difficult to use by itself. However, `groff` provides a *macro* facility to specify how certain routine operations (e.g. starting paragraphs, printing headers and footers, etc.) should be done. These macros can be collected together into a *macro package*. There are a number of macro packages available; the most common (and the ones described in this manual) are `man`, `mdoc`, `me`, `ms`, and `mm`.

1.5 Preprocessors

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

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

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

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

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

Other preprocessors exist, but, unfortunately, no free implementations are available. Among them is a preprocessor for drawing mathematical pictures (`ideal`).

1.6 Output Devices

`groff` produces device-independent code that may be fed into a postprocessor to produce output for a particular device. Currently, `groff` has postprocessors for PostScript devices, character terminals, X11 (for previewing), DVI, HP LaserJet 4 and Canon LBP printers (which use CAPSL), HTML, XHTML, and PDF.

1.7 Credits

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

Larry Kollar contributed the section on 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 them from their original counterparts provided by the host (see Section 2.2 [Environment], page 12). 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 incarnation of troff used. 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 almost all troff implementations[1] (with more or less compatible changes). Similarly, we say 'gpic', 'geqn', and so on.

2.1 Options

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

This section only documents options to the groff front end. Many of the arguments to groff are passed on to gtroff, therefore those are also included. Arguments to pre- or postprocessors can be found in Section 6.3.1 [Invoking gpic], page 199, Section 6.1.1 [Invoking geqn], page 199, Section 6.2.1 [Invoking gtbl], page 199, Section 6.4.1 [Invoking ggrn], page 199, Section 6.7.1 [Invoking grefer], page 199, Section 6.6.1 [Invoking gchem], page 199, Section 6.8.1 [Invoking gsoelim], page 199, Section 6.9.1 [Invoking preconv], page 200, Section 7.2.1 [Invoking grotty], page 201, Section 7.3.1 [Invoking grops], page 202, Section 7.4.1 [Invoking gropdf], page 203, Section 7.8.1 [Invoking grohtml], page 206, Section 7.5.1 [Invoking grodvi], page 204, Section 7.6.1 [Invoking grolj4], page 205, Section 7.7.1 [Invoking grolbp], page 205, and Section 7.9.1 [Invoking gxditview], page 208.

The command-line format for groff is:

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

[1] Besides groff, neatroff is an exception.

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

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

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

Options without an argument can be grouped behind a single `-`. A filename of `-` denotes the standard input. Whitespace is permitted between an option and its argument.

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

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

'`-a`' Generate an ASCII (Unicode basic Latin) approximation of the typeset output. The read-only register `.A` is set to 1. See Section 5.6.5 [Built-in Registers], page 81. On a system using the `man-db` manual page formatter and that installs man pages compressed with `gzip`, one might use the shell command

```
zcat $(man -w troff) | groff -a -t -man -Tdvi \
| less
```

to observe how lines are broken for the DVI device.

'`-b`' Print a backtrace with each warning or error message. This backtrace should help track down the cause of the error. The line numbers given in the backtrace may not always be correct: `gtroff` can get confused by `as` or `am` requests while counting line numbers.

'`-c`' Suppress color output.

'`-C`' Enable compatibility mode. See Section 5.34 [Implementation Differences], page 193, for the list of incompatibilities between `groff` and AT&T `troff`.

'`-dcs`'
'`-dname=s`'
Define *c* or *name* to be a string *s*. *c* must be a one-letter name; *name* can be of arbitrary length. All string assignments happen before loading any macro file (including the start-up file).

'`-Darg`' Set default input encoding used by `preconv` to *arg*. Implies `-k`.

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

'`-E`' Inhibit all error messages.

'`-ffam`' Use *fam* as the default font family. See Section 5.17.2 [Font Families], page 116.

'-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 14). This option is passed to all pre- and postprocessors using the GROFF_FONT_PATH environment variable.

'-g' Preprocess with ggrn.

'-G' Preprocess with grap. Implies -p.

'-h' Print a help message.

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

'-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 6.8 [gsoelim], page 199, for more details); it also implies groff's -s option.
- gtroff; it is used to search files named in the psbb and so requests.
- grops; it is used to search files named in the '\X'ps: import' and '\X'ps: file' escapes.

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

'-j' Preprocess with gchem. Implies -p.

'-k' Preprocess with preconv. This is run before any other preprocessor. Please refer to preconv's manual page for its behaviour if no -K (or -D) option is specified.

'-K*arg*' Set input encoding used by preconv to *arg*. Implies -k.

'-l' Send the output to a spooler for printing. The command used for this is specified by the print command in the device description file (see Section 8.2 [Device and Font Files], page 222). If not present, -l is ignored.

'-L*arg*' Pass *arg* to the spooler. Each argument should be passed with a separate -L option. 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.

'-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).

'-M*dir*' Search directory *dir* for macro files before the standard directories (see Section 2.3 [Macro Directories], page 13).

'`-nnum`' Number the first page *num*.

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

'`-olist`' Output only pages in *list*, which is a comma-separated list of page ranges; '*n*' means print page *n*, '*m*-*n*' means print every page between *m* and *n*, '-*n*' means print every page up to *n*, '*n*-' means print every page beginning with *n*. `gtroff` exits after printing the last page in the list. All the ranges are inclusive on both ends.

Within `gtroff`, this information can be extracted with the '`.P`' register. See Section 5.6.5 [Built-in Registers], page 81.

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

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

'`-Parg`' Pass *arg* to the postprocessor. Each argument should be passed with a separate `-P` option. Note that `groff` does not prepend '`-`' to *arg* before passing it to the postprocessor.

'`-rcn`'
'`-rname=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).

'`-R`' Preprocess with `grefer`. No mechanism is provided for passing arguments to `grefer` because most `grefer` options have equivalent commands that can be included in the file. See Section 6.7 [grefer], page 199, for more details.

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

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

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

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

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

`ps` For PostScript printers and previewers.

`pdf` For PDF viewers or printers.

`dvi` For 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 (ISO 646) character set.

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

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

`cp1047` For typewriter-like devices that use the EBCDIC encoding IBM code page 1047.

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

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

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

The predefined `gtroff` string register `.T` contains the current output device; the read-only number register `.T` is set to 1 if this option is used (which is always true if `groff` is used to call `gtroff`). See Section 5.6.5 [Built-in Registers], page 81.

The postprocessor to be used for a device is specified by the `postpro` command in the device description file. (See Section 8.2 [Device and Font Files], page 222.) This can be overridden with the `-X` option.

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

'`-w`*name*' Enable warning *name*. Available warnings are described in Section 5.33 [Debugging], page 188. Multiple `-w` options are allowed.

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

'`-v`' Make programs run by `groff` print out their version number.

'`-V`' Print the pipeline on `stdout` instead of executing it. If specified more than once, print the pipeline on `stderr` and execute it.

'`-X`' Preview with `gxditview` instead of using the usual postprocessor. This is unlikely to produce good results except with `-Tps`.

This is not the same as using `-TX75` or `-TX100` to view a document with `gxditview`: the former uses the metrics of the specified device, whereas the latter uses X-specific fonts and metrics.

'`-z`' Suppress output from `gtroff`. Only error messages are printed.

'`-Z`' Do not postprocess the output of `gtroff`. Normally `groff` automatically runs the appropriate postprocessor.

2.2 Environment

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

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

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

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

`GROFF_ENCODING`
: The value of this environment value is passed to the `preconv` preprocessor to select the encoding of input files. Setting this option implies `groff`'s command-line option `-k` (that is, `groff` always calls `preconv`). If set without a value, `groff` calls `preconv` without arguments. An explicit `-K` command-line option overrides the value of `GROFF_ENCODING`. See the *preconv(7)* manual page; type `man preconv` at the command line to view it.

`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 14.

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 13.

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.

GROFF_TYPESETTER
: The default output device.

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

MS-DOS and MS-Windows ports of `groff` use semicolons, 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/1.23.0/tmac
  ```

  assuming that the version of `groff` is 1.23.0, 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 116), fonts within a family should start with the family name, followed by the shape. For example, the Times family uses '`T`' for the family name and '`R`', '`B`', '`I`', and '`BI`' to indicate the shapes 'roman', 'bold', 'italic', and 'bold italic', respectively. Thus the final font names are '`TR`', '`TB`', '`TI`', and '`TBI`'.

All font files are kept in the *font directories*, which constitute the *font path*. The file search functions always append the directory `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/1.23.0/font
  ```

  assuming that the version of `groff` is 1.23.0, 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 111, for vertical manipulation of the page size. See Section 5.13 [Line Layout], page 107, 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 222, 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
```

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`'). You can say either '`-m me`' or '`-me`'; the latter is an anachronism from the early days of Unix.[2]

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

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

2.6.1 grog

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

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

For example,

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

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

[2] 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.

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

As this example shows, 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 70.

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

```
.sp
```

spaces one line, but

```
.sp 4
```

spaces four lines. The number 4 is an argument to the `sp` request, which says to space four lines instead of one. Arguments are separated from the request and from each other by spaces (*no* tabs). More details on this can be found in Section 5.5.1.1 [Request and Macro Arguments], page 72.

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 that describe how to format the text. Requests always have a period (‘.’) or an apostrophe (‘’’) as the first character of the input line.

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

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 you to change the way the output is arranged on the page, sometimes called the *layout* of the output page.

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 66).

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 133).

```
.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 that 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 that fall into the following categories.

3.2.1 Paragraphs

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

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

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

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

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

A variation of this is a bulleted list.

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

3.2.2 Sections and Chapters

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

3.2.3 Headers and Footers

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

The titles are called *three-part titles*, that is, there is a left-justified part, a centered part, and a right-justified part. An automatically generated page number may be put in any of these fields with the '`%`' character (see Section 5.15 [Page Layout], page 111, for more details).

3.2.4 Page Layout

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

3.2.5 Displays

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

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

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

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

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

3.2.6 Footnotes and Annotations

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

Footnotes are printed at the bottom of the current page.

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

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

3.2.7 Table of Contents

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

3.2.8 Indices

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

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

3.2.9 Paper Formats

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

3.2.10 Multiple Columns

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

3.2.11 Font and Size Changes

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

3.2.12 Predefined Strings

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

3.2.13 Preprocessor Support

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

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

3.2.14 Configuration and Customization

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

4 Macro Packages

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

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

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

doesn't work. 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`

The `man` macro package is the most widely-used and probably the most important ever developed for `troff`. It is easy to use, and a vast majority of manual pages ("man pages") are written in it.

`groff`'s implementation is documented in the *groff_man(7)* man page. Type `man groff_man` at the command line to view it.

4.1.1 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 typeface (Courier).

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

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

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

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

`.G` [*text*] [Macro]
Set *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]
Set *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]
Set *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 man page 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 a constant-width typeface (Courier), followed by optional punctuation.

`.Pn` [*punct*] *path* [*punct*] [Macro]
If called with two arguments, identical to `PN`. If called with three arguments, set the second argument in a constant-width typeface (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 space in front of headings.
.rn SH SH-orig
.de SH
.  if t .sp (u;\\n[PD]*2)
.  SH-orig \\$*
..
```

4.2 mdoc

`groff`'s implementation of the BSD `doc` package for man pages is documented in the *groff_mdoc(7)* man page. Type `man groff_mdoc` at the command line to view it.

4.3 me

`groff`'s implementation of the BSD `me` macro package is documented using itself. A tutorial, `meintro.me`, and reference, `meref.me`, are available in `groff`'s documentation directory. A *groff_me(7)* man page is also available and identifies the installation path for these documents. Type `man groff_me` at the command line to view it.

A French translation of the tutorial is available as `meintro_fr.me` and installed parallel to the English version.

4.4 mm

`groff`'s implementation of the AT&T memorandum macro package is documented in the *groff_mm(7)* man page. Type `man groff_mm` at the command line) to view it.

A Swedish localization of `mm` is also available; see *groff_mmse(7)*.

4.5 mom

The main documentation files for the `mom` macros are in HTML format. Additional, useful documentation is in PDF format. See the *groff(1)* man page, section "Installation Directories", for their location.

- `toc.html` Entry point to the full mom manual.
- `macrolist.html` Hyperlinked index of macros with brief descriptions, arranged by category.
- `mom-pdf.pdf` PDF features and usage.

The mom macros are in active development between groff releases. The most recent version, along with up-to-date documentation, is available at `http://www.schaffter.ca/mom/mom-05.html`.

The *groff_mom(7)* man page (type `man groff_mom` at the command line) contains a partial list of available macros, however their usage is best understood by consulting the HTML documentation.

4.6 ms

The `ms` ("manuscript") macros are suitable for reports, letters, memoranda, books, user manuals, and so forth. The package provides macros for cover page and table of contents generation, section headings, multiple paragraph styles, text styling (including font changes), lists, footnotes, pagination, and indexing.

`ms` supports the `tbl`, `eqn`, `pic`, and `refer` preprocessors for inclusion of tables, mathematical equations, diagrams, and standardized bibliographic citations.

4.6.1 Introduction to ms

The `ms` macros are the oldest surviving macro package for `roff` systems.[1] While the `man` package was intended for brief documents to be perused at a terminal, the `ms` macros are suitable for longer documents intended for printing and possible publication.

The `ms` macro package included with `groff` is a complete reimplementation. Some macros specific to AT&T or Berkeley are not

[1] Although man *pages* are even older, the `man` macro language dates back only to Seventh Edition Unix (1979). `ms` was documented by Mike Lesk in an article for the *Communications of the ACM* in 1974.

included, while several new commands been introduced. See Section 4.6.7 [Differences from AT&T ms], page 50.

If you're in a hurry to get started, you need only know that `ms` needs one of its macros called at the beginning of a document so that it can initialize. A paragraph macro like `PP` (if you want your paragraph to have a first-line indent) or `LP` (if you don't) suffices.

After that, start typing normally. You can separate paragraphs with further paragraph macros, or with blank lines, and you can indent with tabs. When you need one of the features mentioned earlier (see Section 4.6 [ms], page 26), return to this manual.

```
.LP
Radical novelties are so disturbing that they tend to be
suppressed or ignored, to the extent that even the
possibility of their existence in general is more often
denied than admitted.

→That's what Dijkstra said, anyway.
```

We have used an arrow → in the above to indicate a tab character.

4.6.2 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, `ms` prints the cover page information on its own page; otherwise it prints the information (if any) on the first page with your document text immediately following. Some document types found in AT&T `troff` are specific to AT&T or Berkeley, and are not supported in `groff`.

Format and layout
: By setting registers (and one string), you can change your document's type (font and point size), margins, spacing, headers and footers, and footnotes. See Section 4.6.3 [ms Document Control Settings], page 28.

Cover page
: A cover page consists of a title, the author's name and institution, an abstract, and the date.[2] See Section 4.6.4 [ms Cover Page Macros], page 32.

Body
: Following the cover page is your document. `ms` supports highly structured documents consisting of paragraphs interspersed with multi-level headings (chapters, sections, subsections, and so forth) and augmented by lists, footnotes, tables, diagrams, and similar. See Section 4.6.5 [ms Body Text], page 34.

Table of contents
: Longer documents usually include a table of contents, which you can produce by placing the `TC` macro at the end of your document. Printing the table of contents at the end is necessary since GNU `troff`, like its AT&T ancestor, is a single-pass text formatter; it thus cannot determine the page number of each section until that section has been set and output. Since `ms` output is designed for hard copy, you can manually relocate the pages containing the table of contents between the cover page and the body text after printing.[3]

4.6.3 Document control settings

`ms` exposes many aspects of document layout to user control via `groff` requests. To use them, you must understand how to define registers and strings.

`.nr` *reg value* [Request]
Set register *reg* to *value*. If *reg* doesn't exist, GNU `troff` creates it.

`.ds` *name contents* [Request]
Set string *name* to *contents*. If *name* exists, it is removed first.

For 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.

A list of document control registers (and one string) follows. They are presented in the syntax used to interpolate them.

---

[2] Actually, only the title is required.

[3] This limitation could also be overcome by using PostScript or PDF file manipulation utilities to resequence pages in the document, facilitated by specially-formatted comments ("device tags") placed in the output by by `ms`.

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.

Effective: next paragraph.

Default: 12 p.

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

Effective: next section heading.

Default: 1 p.

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

Effective: next section heading.

Default: 0.

`\n[HY]` [Register]
Defines the hyphenation mode. `HY` safely sets the value of the low-level `hy` register. Setting `HY` to 0 is equivalent to using the `nh` request.

Effective: next paragraph.

Default: 6.

`\*[FAM]` [String]
Defines the font family used to typeset the document.

Unlike the other document control settings, `FAM` is a string, not a register. You must therefore set it with the `ds` request instead of `nr`.

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`, `QS`, and `QE` macros) paragraph.

Effective: next paragraph.

Default: 5 n.

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

Effective: next paragraph.

Default: 1.

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

Effective: next paragraph.

Default: 1.

Footnote Settings

`\n[FL]` [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 Registers

`\n[MINGW]` [Register]

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

Effective: next page.

Default: 2 n.

`\n[DD]` [Register]

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

Effective: next paragraph.

Default: 0.5 v.

4.6.4 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 ...
```

4.6.5 Body text

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

4.6.5.1 Paragraphs

The following paragraph types are available.

`.PP` [Macro]
Sets a paragraph with an initial indentation.

`.LP` [Macro]
Sets a paragraph without an initial indentation.

`.QP` [Macro]
Sets a paragraph that is indented at both left and right margins by the amount of the register `QI`. The next paragraph or heading returns margins

to normal. QP inserts vertical space of amount set by register PD before the paragraph.

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

.XP [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 4.6.3 [ms Document Control Settings], page 28) operates in conjunction with each of these macros, to inhibit the printing of orphan lines at the bottom of any page.

4.6.5.2 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 ms prints a warning on the standard error stream.

*[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 (as it appears in a printed section heading with default formatting, followed by a terminating period), and SN-NO-DOT (with the terminating period omitted). The string SN is also defined, as an alias for SN-DOT; if preferred, you may redefine it as an alias for SN-NO-DOT, by including the initialization

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

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

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

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

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

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

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

.SH [*match-level*] [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 4.6.3 [ms Document Control Settings], page 28.

The `HORPHANS` register (see Section 4.6.3 [ms Document Control Settings], page 28) operates in conjunction with the `NH` and `SH` macros, to inhibit the printing of orphaned section headings at the bottom of any page.

4.6.5.3 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 ms` 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 ms` 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.

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

`.CW` [*txt* [*post* [*pre*]]] [Macro]
Sets its first argument in a `constant-width typeface`. It operates similarly to the `B` macro otherwise. This is a Berkeley extension.

In `groff ms` you might prefer to change the font family to Courier—a constant-width typeface—by setting the `FAM` string to '`C`'. You can then use all four style macros above, returning to the default family (Times) with '`.ds FAM T`'.

`.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.

`\*[<]` [String]
`\*[>]` [String]
Text enclosed with `\*<` and `\*>` is printed as a subscript.

4.6.5.4 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 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 4.6.3 [ms Document Control Settings], page 28) 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 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 4.6.5.5 [Indentation values in ms], page 41.

For example:

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

Produces:

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

4.6.5.5 Indentation values

In many situations, you may need to indentation a section of text while still wrapping and filling. See Section 4.6.5.4 [Lists in ms], page 38, 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 4.6.5.7 [ms Displays and Keeps], page 42, for macros to indentation and turn off filling.

4.6.5.6 Tab Stops

Use the `ta` request to define tab stops as needed. See Section 5.10 [Tabs and Fields], page 97.

`.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.

4.6.5.7 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.

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).

4.6.5.8 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.

4.6.5.9 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
```

4.6.5.10 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 4.6.3 [ms Document Control Settings], page 28.

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 `\**`.

4.6.6 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 registers and strings.

4.6.6.1 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.

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

`.PT` [Macro]
`.HD` [Macro]

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

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

4.6.6.2 Margins

You control margins using a set of registers. See Section 4.6.3 [ms Document Control Settings], page 28, for details.

4.6.6.3 Multiple columns

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

`.1C` [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.

4.6.6.4 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.

4.6.6.5 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[4]:

`\*[-]` [String]
Prints an em dash.

[4] For an explanation what special characters are see Section 7.1 [Special Characters], page 201.

`\*[Q]` [String]
`\*[U]` [String]
Prints typographer's quotes where available, and neutral quotes otherwise. `\*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.

`\*[`]` [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.

4.6.7 Differences from AT&T `ms`

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

- The internals of `groff ms` differ from the internals of AT&T '`troff -ms`'. Documents that depend upon implementation details of AT&T `troff ms` may not format properly with `groff ms`.
- The general error-handling policy of `groff ms` is to detect and report errors, rather than silently to ignore them.
- `groff ms` does not work in compatibility mode (that is, with the `-C` option).
- There is no special support for terminal 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` registers can be used to control the column width and gutter width, respectively. These 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: accessing an undefined register automatically defines it.
- Displays are left-adjusted by default, not indented. In AT&T `troff ms`, '`.DS`' is synonymous with '`.DS I`'; in `groff ms`, it is synonymous with '`.DS L`'.
- Right-adjusted displays are available. The AT&T `troff ms` manual observes that "it is tempting to assume that '`.DS R`' will right adjust lines, but it doesn't work".[5] In `groff ms`, it does.

`\n[GS]` [Register]
: This 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 register.

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

These are not enabled by default because (a) they were not documented in the original `ms` manual[6] and (b) the `P1` and `UC` macros collide with different macros with the same names in the Berkeley version of `ms`.

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

4.6.7.1 `troff` macros not appearing in `groff`

Macros missing from `groff ms` are 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

[5] "Typing Documents on the UNIX System: Using the -ms Macros with Troff and Nroff"; M. E. Lesk; Bell Laboratories; 1978.

[6] *Ibid.*

`.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 Technical Report; a cover sheet style
`.OK`	Other keywords
`.CS`	Cover sheet information
`.MH`	Murray Hill Bell Laboratories postal address

4.6.7.2 `groff` macros not appearing in AT&T `troff`

The `groff ms` macros have a few minor extensions to the AT&T '`troff -ms`' macros.

`.AM` [Macro]
Use improved accent marks. See Section 4.6.6.5 [ms Strings and Special Characters], page 48, for details. This is a Berkeley extension.

`.CW` [Macro]
Set text in a `constant-width` font (Courier). This is a Berkeley extension.

`.IX` [Macro]
Write an indexing term to the standard error stream. You can write a script to capture and process an index generated in this manner.

The following additional registers appear in `groff ms`.

`\n[MINGW]` [Register]
Specifies a minimum space ("gutter width") between columns (for multi-column output); this takes the place of the `GW` register that was introduced in the Seventh Edition Unix (1979) version of the AT&T '`troff -ms`' macros.

Several new strings are available as well. You can change these to handle (for example) the local language. See Section 4.6.6.5 [ms Strings and Special Characters], page 48, for details.

4.6.8 ms Naming Conventions

The following conventions are used for names of macros, strings, and 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. The 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.

5 gtroff Reference

This chapter covers *all* of the facilities of the GNU `troff` formatting engine. Users of macro packages may skip it if not interested in details.

5.1 Text

AT&T `troff` was designed to take input as it would be composed on a typewriter, including the teletypewriters used as early computer terminals, and relieve the user of having to be concerned with the precise line length that the final version of the document would use, where words should be hyphenated, and how to achieve straight margins on both the left and right sides of the page. Early in its development, the program gained the ability to prepare output for a phototypesetter; a document could then be prepared for output to either a teletypewriter, a phototypesetter, or both. GNU `troff` continues this tradition of permitting an author to compose a single master version of a document which can then be rendered for a variety of output formats or devices.

GNU `troff` input files contain text with directives to control the typesetter interspersed throughout. Even in the absence of such directives, GNU `troff` still processes its input in several ways, by filling, hyphenating, breaking, and adjusting it.

5.1.1 Filling

When GNU `troff` starts up, it obtains information about the device for which it is preparing output.[1] A crucial example is the length of the output line, such as "6.5 inches".

GNU `troff` processes its input by reading words. To GNU `troff`, a *word* is any sequence of one or more characters that aren't spaces, tabs, or newlines. They are separated by spaces, tabs, newlines, or file boundaries.[2] GNU `troff` reads its input character by character, collecting words as it goes, and fits as many of them together on one output line as it can—this is known as *filling*.

```
It is a truth universally acknowledged
that a single man in possession of a
good fortune must be in want of a wife.
    ⇒ It is a truth universally acknowledged that a
    ⇒ single man in possession of a good fortune must
    ⇒ be in want of a wife.
```

[1] Section 8.2 [Device and Font Files], page 222.

[2] There are also *escape sequences* which can function as word characters, word-separating space, or neither—the last simply have no effect on GNU `troff`'s idea of whether its input is within a word or not.

5.1.2 Sentences

A passionate debate has raged for decades among writers of the English language over whether more space should appear between adjacent sentences than between words within a sentence, and if so, how much, and what other circumstances should influence this spacing.[3] GNU `troff` follows the example of AT&T `troff`, attempting to detect the boundaries between sentences, and supplying additional inter-sentence space.

```
Hello, world!
Welcome to groff.
    ⇒ Hello, world!  Welcome to groff.
```

GNU `troff` does this by flagging certain characters (normally '`!`', '`?`', and '`.`') as *end-of-sentence* characters; when GNU `troff` encounters one of these characters at the end of a line, or one of them is followed by two or more spaces on the same input line, it appends a normal space followed by an inter-sentence space in the formatted output.

```
R. Harper subscribes to a maxim of P. T. Barnum.
    ⇒ R. Harper subscribes to a maxim of P. T. Barnum.
```

In the above example, inter-sentence space is not added after '`P.`' or '`T.`' because the periods do not occur at the end of an input line, nor are they followed by two or more spaces. Let's imagine that we've heard something about defamation from Mr. Harper's attorney, recast the sentence, and reflowed it in our text editor.

```
I submit that R. Harper subscribes to a maxim of P. T.
Barnum.
    ⇒ I submit that R. Harper subscribes to a maxim of
    ⇒ P. T.  Barnum.
```

"Barnum" doesn't begin a sentence! What to do? Let us meet our first *escape sequence*, a series of input characters that give special instructions to GNU `troff` instead of being copied as-is to output device glyphs.[4] An escape sequence begins with the backslash character `\` by default, an uncommon character in natural language text, and is *always* followed by at least one other character, hence the term "sequence".

The non-printing input break escape sequence `\&` can be used after an end-of-sentence character to defeat end-of-sentence detection on a per-instance basis. We can therefore rewrite our input more defensively.

---

[3] A well-researched jeremiad appreciated by `groff` contributors on both sides of the sentence-spacing debate can be found at `https://web.archive.org/web/20171217060354/http://www.heracliteanriver.com/?p=324`.

[4] This statement oversimplifes; there are escape sequences whose purpose is precisely to produce glyphs on the output device, and input characters that *aren't* part of escape sequences can undergo a great deal of processing before getting to the output.

```
I submit that R. Harper subscribes to a maxim of P.\& T.\&
Barnum.
    ⇒ I submit that R. Harper subscribes to a maxim of
    ⇒ P. T. Barnum.
```

Was the additional `\&` after '`P.`' necessary? No, but what if further editing and reflowing places '`P.`' at the end of an input line? Ensuring that sentence boundaries are robust to editing activities and reliably understood both by GNU `troff` and the document author is a goal of the advice presented in Section 5.1.9 [Input Conventions], page 63.

Normally, the occurrence of a visible non-end-of-sentence character (as opposed to a space or tab) after an end-of-sentence character cancels detection of the end of a sentence. For example, it would be incorrect for GNU `troff` to infer the end of a sentence after the dot in '`3.14159`'. However, several characters are treated *transparently* after the occurence of an end-of-sentence character. That is, GNU `troff` does not cancel the end-of-sentence detection process when it processes them. This is because such characters are often used as footnote markers or to close quotations and parentheticals. The default set is '`"`', '`'`', '`)`', '`]`', '`*`', `\[dg]`, `\[dd]`, `\[rq]`, and `\[cq]`. The last four are examples of *special characters*, escape sequences whose purpose is to obtain glyphs that are not easily typed at the keyboard, or which have special meaning to GNU `troff` (like `\` itself).

```
\[lq]The idea that the poor should have leisure has always
been shocking to the rich.\[rq]
(Bertrand Russell, 1935)
    ⇒ "The idea that the poor should have
    ⇒ leisure has always been shocking to
    ⇒ the rich."  (Bertrand Russell, 1935)
```

The sets of characters that potentially end sentences or are transparent to sentence endings are configurable. See the `cflags` request in Section 5.17.4 [Using Symbols], page 119. To change the additional inter-sentence spacing amount—even to remove it entirely—see the `ss` request in Section 5.7 [Manipulating Filling and Adjustment], page 83.

5.1.3 Hyphenation

It is uncommon for the most recent word collected from the input to exactly fill the output line. Typically, there is enough room left over for part of the next word. The process of splitting a word so that it appears partially on one line (with a hyphen to indicate to the reader that the word has been broken) and the remainder of the word on the next is *hyphenation*. GNU `troff` uses a hyphenation algorithm and language-specific pattern files (based on but simplified from those used in TEX) to decide which words can be hyphenated and where.

Hyphenation does not always occur even when the hyphenation rules for a word allow it; it can be disabled, and when not disabled there are several

parameters that can prevent it. See Section 5.8 [Manipulating Hyphenation], page 88.

5.1.4 Breaking

Once an output line has been filled, whether or not hyphenation has occurred on that line, the next word read from the input will be placed on a different output line; this is called a *break*. In this manual and in `roff` discussions generally, a "break" if not further qualified always refers to the termination of an output line. After an automatic break, GNU `troff` adjusts the line if applicable (see below), and then resumes collecting and filling text on the next output line.

Sometimes, a line cannot be broken automatically. This typically does not happen with natural language text unless the output line length has been manipulated to be extremely short, but it can with specialized text like program source code. We can use `perl` at the shell prompt to contrive an example of failure to break the output line. The regular output is omitted below.

```
$ perl -e 'print "\$" x 80, "\n";' | nroff
     error  troff: <standard input>:1: warning [p 1, 0.0i]:
     error  can't break line
```

The remedy for these cases is to tell GNU `troff` where the line may be broken without hyphens. This is done with the non-printing break point escape sequence; see Section 5.8 [Manipulating Hyphenation], page 88.

What if the document author wants to stop filling lines temporarily, for instance to start a new paragraph? There are several solutions. A blank line not only causes a break, but by default it also outputs a one-line vertical space (effectively a blank line). This behavior can be modified with the blank line macro request `blm`. See Section 5.24.4 [Blank Line Traps], page 167. Macro packages may discourage or disable the blank line method of paragraphing in favor of their own macros.

A line that begins with a space causes a break and the space is output at the beginning of the next line. This space isn't *adjusted* (see below); however, this behavior can be modified with the leading spaces macro request `lsm`. See Section 5.24.5 [Leading Spaces Traps], page 167. Again, macro packages may provide other methods of producing indented paragraphs.

What if there is no next input word? Or the file ends before enough words have been collected to fill an output line? The end of the file also causes a break, resolving both of these cases. Certain requests also cause breaks, implicitly or explicitly. This is discussed in Section 5.7 [Manipulating Filling and Adjustment], page 83.

5.1.5 Adjustment

Once GNU `troff` has filled a line and performed an automatic break, it tries to *adjust* that line; additional inter-sentence space is inserted (and, in

the default adjustment mode, inter-word spaces are widened until the text reaches the right margin). Extra spaces between words are preserved, but trailing spaces on an input line are ignored. Leading spaces are handled as noted above. Text can be adjusted to the left or right margins only (instead of both), or centered; see Section 5.7 [Manipulating Filling and Adjustment], page 83. As a rule, an output line that has not been filled will not be adjusted.

5.1.6 Tab Stops

GNU `troff` translates horizontal tab characters, also called simply "tabs", in the input into movements to the next tab stop. These tab stops are by default located every half inch across the page. With them, simple tables can be made easily.[5] However, this method can be deceptive as the appearance (and width) of the text on a terminal and the results from GNU `troff` can vary greatly, particularly when proportional typefaces are used.

A further possible difficulty is that lines beginning with tab characters are still filled, possibly producing unexpected results.[6]

```
1       2       3
        4       5
```

The above example produces the following output.

```
1       2       3                       4       5
```

GNU `troff` provides sufficient facilities for sophisticated table composition; Section 5.10 [Tabs and Fields], page 97. There are many details to track when using such low-level features, so most users turn to the *tbl*(1) preprocessor (type `man tbl` at the command line) for table construction.

5.1.7 Requests and Macros

We have now encountered almost all of the syntax there is in `roff` languages, with one conspicuous exception.

A *request* is an instruction to the formatter that occurs on a line by itself after a control character.[7] A *control character* must occur at the beginning of an input line to be recognized. The regular control character has a counterpart, the *no-break control character*, which suppresses the break that is implied by some requests. The default control characters are the dot (`.`) and the neutral apostrophe (`'`), the latter being the no-break control character. These characters were chosen because it is uncommon for lines of text in natural languages to begin with periods or apostrophes.

GNU `troff` requests, combined with its escape sequences, comprise the control language of the formatter. Of key importance are the requests that

[5] "Tab" is short for "tabulation", revealing the term's origin as a spacing mechanism for table arrangement.

[6] It works well, on the other hand, for a traditional practice of paragraph composition wherein a tab is used to create a first-line indentation.

[7] Or occasionally as part of another request, such as `if` or `while`.

define macros. Macros are invoked like requests, enabling the request repertoire to be extended or overridden.[8]

A macro can be thought of as an abbreviation that is automatically replaced with what it stands for. In `roff` systems, the process of replacing a macro is known as *interpolation*.[9] Interpolations are handled as soon as they are recognized, and once performed, a `roff` formatter scans the replacement for further requests, macro calls, and escape sequences.

In `roff` systems, the `de` request defines a macro.[10]

```
.de DATE
2020-11-14
..
```

The foregoing input produces no output by itself; all we have done is store some information. Observe the pair of dots that end the macro definition. This is a default; you can specify your own terminator for the macro definition.

```
.de NAME ENDNAME
Heywood Jabuzzoff
.ENDNAME
```

In fact, the ending marker is no mere string, but can itself be a macro that will be automatically called if it is defined at the time the enclosing macro definition begins.

```
.de END
Big Rip
..
.de START END
Big Bang
.END
.START
    ⇒ Big Rip Big Bang
```

In the foregoing example, "Big Rip" printed before "Big Bang" because its macro was *called* first. Consider what would happen if we dropped `END` from the '`.de START`' line and added `..` after `.END`. Would the order change?

Macro definitions can be collected into *macro packages*, `roff` input files designed to produce no output themselves but instead ease the preparation of other `roff` documents. Macro packages can be loaded by supplying the `-m` option to `groff` or `troff`. Alternatively, a `groff` document wishing to use a macro package can load it with the `mso` ("macro source") request.

---

[8] Argument handling in macros is more flexible but also more complex. See Section 5.5.1.1 [Request and Macro Arguments], page 72.

[9] Some escape sequences undergo interpolation as well.

[10] GNU `troff` offers several others. See Section 5.21 [Writing Macros], page 148.

```
.de DATE
2020-10-05
..
.
.de BOSS
D.\& Kruger,
J.\& Peterman
..
.
.de NOTICE
Approved:
.DATE
by
.BOSS
..
.
Insert tedious regulatory compliance paragraph here.

.NOTICE

Insert tedious liability disclaimer paragraph here.

.NOTICE
    ⇒ Insert tedious regulatory compliance paragraph here.
    ⇒
    ⇒ Approved: 2020-10-05 by D. Kruger, J. Peterman
    ⇒
    ⇒ Insert tedious liability disclaimer paragraph here.
    ⇒
    ⇒ Approved: 2020-10-05 by D. Kruger, J. Peterman
```

The document started with a series of lines beginning with the control character. Three macros were defined, with a `de` request declaring the macro's name, and the "body" of the macro starting on the next line and continuing until a line with two dots '`..`' marked its end. The text proper began only after the macros were defined; this is a common pattern. Only the `NOTICE` macro was called "directly" by the document; `DATE` and `BOSS` were called only by `NOTICE` itself. Escape sequences were used in `BOSS`, two levels of macro interpolation deep.

The advantage in typing and maintenance economy may not be obvious from such a short example, but imagine a much longer document with dozens of such paragraphs, each requiring a notice of managerial approval. Consider what must happen if you are in charge of generating a new version of such a document with a different date, for a different boss. With well-chosen macros, you only have to change each datum in one place.

In practice, we would probably use strings (see Section 5.19 [Strings], page 137) instead of macros for such simple interpolations; what is important here is to glimpse the potential of macros and the power of recursive interpolation.

We could have defined `DATE` and `BOSS` in the opposite order; perhaps less obviously, we could also have defined them *after* `NOTICE`. "Forward references" like this are acceptable because the body of a macro definition is not (completely) interpreted, but stored instead (see Section 5.21.1 [Copy Mode], page 151). While a macro is being defined, requests are not interpreted and macros not interpolated, whereas some commonly used escape sequences *are* interpolated. `roff` systems also support mutually recursive macros—as long as you have a way to break the recursion (see Section 5.20 [Conditionals and Loops], page 143). For maintainable `roff` documents, arrange your macro definitions so that they are most easily understood when read from beginning to end.

5.1.8 Input Encodings

The `groff` front end calls the `preconv` preprocessor to handle most input character encoding issues without troubling the user. Direct input to GNU `troff`, on the other hand, must be in one of two encodings it can recognize.

cp1047 The code page 1047 input encoding works only on EBCDIC platforms (and conversely, the other input encodings don't work with EBCDIC); the file `cp1047.tmac` is by default loaded at start-up.

latin1 ISO Latin-1, an encoding for Western European languages, is the default input encoding on non-EBCDIC platforms; the file `latin1.tmac` is loaded at start-up.

Any document that is encoded in ISO 646:1991 (a descendant of USAS X3.4-1968 or "US-ASCII"), or, equivalently, uses only code points from the "C0 Controls" and "Basic Latin" parts of the Unicode character set is also a valid ISO Latin-1 document; the standards are interchangeable in their first 128 code points.[11]

The remaining encodings require support that is not built-in to the GNU `troff` executable; instead, they use macro packages.

latin2 To use ISO Latin-2, an encoding for Central and Eastern European languages, either use '`.mso latin2.tmac`' at the very beginning of your document or use '`-mlatin2`' as a command-line argument to `groff`.

latin5 To use ISO Latin-5, an encoding for the Turkish language, either use '`.mso latin5.tmac`' at the very beginning of your document or use '`-mlatin5`' as a command-line argument to `groff`.

[11] The *semantics* of certain punctuation code points have gotten stricter with the successive standards, a cause of some frustration among man page writers; see the *groff_char(7)* man page.

latin9 ISO Latin-9 is intended (at least in Europe) to replace Latin-1. Its main difference from Latin-1 is that Latin-9 contains the Euro character. To use this encoding, either use '`.mso latin9.tmac`' at the very beginning of your document or use '`-mlatin9`' as a command-line argument to `groff`.

Some input encoding characters may not be available for a particular output device.

```
groff -Tlatin1 -mlatin9 ...
```

The above command fails if you use the Euro character in the input. Usually, this limitation is present only for devices that have a limited repertoire of output glyphs (e.g., `-Tascii` and `-Tlatin1`); for other devices it is usually sufficient to install proper fonts that contain the necessary glyphs.

Due to the importance of the Euro glyph in Europe, `groff` is distributed with a PostScript font called `freeeuro.pfa`, which provides various glyph shapes for the Euro. In other words, Latin-9 encoding is supported for the `-Tps` device out of the box (Latin-2 isn't).

The `-Tutf8` device supports characters from all other 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).

5.1.9 Input Conventions

Since GNU `troff` fills text automatically, it is common practice in `roff` languages to not attempt careful visual composition of text in input files: it is the esthetic appeal of the formatted output that matters. Instead, `troff` input should be arranged such that it is easy for authors and maintainers to compose and develop the document, understand the syntax of `roff` requests, macro calls, and preprocessor languages used, and predict the behavior of the formatter. Several traditions have accrued in service of these goals.

- Break input lines after sentence-ending punctuation to ease their recognition (see Section 5.1.2 [Sentences], page 56). It is frequently convenient to break after colons and semicolons as well, as these typically precede independent clauses. Consider breaking after commas; they often occur in lists that become easy to scan when itemized by line, or constitute supplements to the sentence that are added, deleted, or updated to clarify it. Parenthetical and quoted phrases are also good candidates for placement on input lines by themselves.
- Set your text editor's line length to 72 characters or fewer.[12] This limit, combined with the previous advice regarding breaking around punctuation, makes it less common that an input line will wrap in your text editor, and thus will help you perceive excessively long constructions in your text. Recall that natural languages originate in speech, not writ-

[12] Emacs: `fill-column: 72`; Vim: `textwidth=72`

ing, and that punctuation is correlated with pauses for breathing and changes in prosody.

- Use `\&` after '`!`', '`?`', and '`.`' if they are followed by space or tab characters and don't end a sentence.
- Do not attempt to format the input in a WYSIWYG manner (i.e., don't try using spaces to get proper indentation or align columns of a table).
- Comment your document. It is never too soon to apply comments to record information of use to future document maintainers (including your future self). We thus introduce another escape sequence, `\"`, which causes GNU `troff` to ignore the remainder of the input line.
- Use the empty request to visually manage separation of material in input files. The `groff` project's own documents use an empty request between sentences and after macro definitions, and two empty requests between paragraphs or other requests or macro calls that will introduce vertical space into the document.

  Combined with the comment escape, you can include whole-line comments in your document, and even "comment out" sections of your document by prefixing lines with empty requests and the comment escape.

An example sufficiently long to illustrate the above suggestions in practice follows. For the purpose of fitting the example in the margins of this manual with the font used for its typeset version, we have shortened the input line length to 58 columns. We have also used an arrow → to indicate a tab character.

```
.\" raw roff input example
.\"   nroff this_file.roff | less
.\"   groff this_file.roff > this_file.ps
→The theory of relativity is intimately connected with the
theory of space and time.
.
I shall therefore begin with a brief investigation of the
origin of our ideas of space and time,
although in doing so I know that I introduce a
controversial subject.
.
.\" remainder of paragraph elided
.
.

→The experiences of an individual appear to us arranged in
a series of events;
in this series the single events which we remember appear
to be ordered according to the criterion of
\[lq]earlier\[rq] and \[lq]later\[rq], \" punct swapped
which cannot be analysed further.
.
There exists,
therefore,
for the individual,
an I-time,
or subjective time.
.
This itself is not measurable.
.
I can,
indeed,
associate numbers with the events,
in such a way that the greater number is associated with
the later event than with an earlier one;
but the nature of this association may be quite arbitrary.
.
This association I can define by means of a clock by
comparing the order of events furnished by the clock with
the order of a given series of events.
.
We understand by a clock something which provides a series
of events which can be counted,
and which has other properties of which we shall speak
later.
.\" Albert Einstein, _The Meaning of Relativity_, 1922
```

5.2 Measurements

`gtroff` (like many other programs) requires numeric parameters to specify various measurements. Most numeric parameters[13] may have a *measurement unit* attached. These units are specified as a single character that immediately follows the number or expression. Each of these units are understood, by `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 independent of any of the current settings (e.g., type size) of GNU `troff`.

Although GNU `troff`'s basic unit is device-dependent, it may still be smaller than the smallest unit the device is capable of producing. The register `.H` specifies how many groff basic units constitute the current device's basic unit horizontally, and the register `.V` specifies this value vertically.

`i` Inches. An antiquated measurement unit still in use in certain backwards countries with incredibly low-cost computer equipment. One inch is defined to be 2.54 cm (worldwide since 1964).

`c` Centimeters. One centimeter is about 0.3937 in.

`p` Points. This is a typesetter's measurement used for measure type size. It is 72 points to an inch.

`P` Pica. Another typesetting measurement. 6 picas to an inch (and 12 points to a pica).

`s`
`z` See Section 5.18.2 [Fractional Type Sizes], page 136, for a discussion of these units.

`f` Fractions. Value is 65536. See Section 5.28 [Colors], page 177, for usage.

The other measurements understood by `gtroff` depend on settings currently in effect in `gtroff`. These are very useful for specifying measurements that should look proper with any size of text.

`m` Ems. This unit is equal to the current font size in points. So called because it is *approximately* the width of the letter '`m`' in the current font.

`n` Ens. In `groff`, this is half of an em.

`v` Vertical space. This is equivalent to the current line spacing. See Section 5.18 [Sizes], page 133.

`M` 100ths of an em.

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

5.2.1 Default Units

Many requests take a default unit. While this can be helpful at times, it can cause strange errors in some expressions. For example, the line length request expects em units. Here are several attempts to get a line length of 3.5 inches and their results:

```
3.5i       ⇒   3.5i
7/2        ⇒   0i
7/2i       ⇒   0i
(7 / 2)u   ⇒   0i
7i/2       ⇒   0.1i
7i/2u      ⇒   3.5i
```

Everything is converted to basic units first. In the above example it is assumed that 1 i equals 240 u, and 1 m equals 10 p (thus 1 m equals 33 u). The value 7 i/2 is first handled as 7 i/2 m, then converted to 1680 u/66 u, which is 25 u, and this is approximately 0.1 i. As can be seen, a scaling indicator after a closing parenthesis is simply ignored.

Thus, the safest way to specify measurements is to always attach a scaling indicator. If you want to multiply or divide by a certain scalar value, use '`u`' as the unit for that value.

5.3 Expressions

`gtroff` has most arithmetic operators common to other languages:

- Arithmetic: '`+`' (addition), '`-`' (subtraction), '`/`' (division), '`*`' (multiplication), '`%`' (modulo).

  `gtroff` only provides integer arithmetic. The internal type used for computing results is '`int`', which is usually a 32-bit signed integer.

- Comparison: '`<`' (less than), '`>`' (greater than), '`<=`' (less than or equal), '`>=`' (greater than or equal), '`=`' (equal), '`==`' (the same as '`=`').

- Logical: '`&`' (logical and), '`:`' (logical or).

- Unary operators: '`-`' (negating, i.e., changing the sign), '`+`' (just for completeness; does nothing in expressions), '`!`' (logical not; this works only within `if` and `while` requests).[14] See below for the use of unary operators in motion requests.

  The logical not operator, as described above, works only within `if` and `while` requests. Furthermore, it may appear only at the beginning of an expression, and negates the entire expression. Attempting to insert the '`!`' operator within the expression results in a '`numeric expression expected`' warning. This maintains compatibility with AT&T `troff`.

  Example:

[14] For example, '`!(-1)`' evaluates to 'true' because GNU `troff` treats both negative numbers and zero as 'false'.

```
.nr X 1
.nr Y 0
.\" This does not work as expected.
.if (\n[X])&(!\n[Y]) .nop X only
.
.\" Use this construct instead.
.if (\n[X]=1)&(\n[Y]=0) .nop X only
```

- Extrema: '`>?`' (maximum), '`<?`' (minimum).

  Example:

  ```
  .nr x 5
  .nr y 3
  .nr z (\n[x] >? \n[y])
  ```

  The register `z` now contains 5.

- Scaling: `(c;e)`. Evaluate *e* using *c* as the default scaling indicator. If *c* is missing, ignore scaling indicators in the evaluation of *e*.

Parentheses may be used as in any other language. However, in `gtroff` they are necessary to ensure order of evaluation. `gtroff` has no operator precedence; expressions are evaluated left to right. This means that `gtroff` evaluates '`3+5*4`' as if it were parenthesized like '`(3+5)*4`', not as '`3+(5*4)`', as might be expected.

For many requests that cause a motion on the page, the unary operators '`+`' and '`-`' work differently if leading an expression. They then indicate a motion relative to the current 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 76, 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 72, and Section 5.20 [Conditionals and Loops], page 143.

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 the ISO 646, 8859, or 10646 character encodings, causing a warning message of type 'input' (see Section 5.33 [Debugging], page 188, 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 GNU troff to cover Unicode or other character sets and encodings that use characters of these ranges.[15]

  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
@_
```

An identifier longer than two characters with a closing bracket (']') in its name can't be accessed with escape sequences that expect an identifier as a parameter. For example, '\[foo]]' accesses the glyph 'foo', followed by ']', whereas '\C'foo]'' really asks for glyph 'foo]'.

To avoid problems with the refer preprocessor, macro names should not start with '[' or ']'. Due to backwards compatibility, everything after '.[' and '.]' is handled as a special argument to refer. For example, '.[foo' makes refer to start a reference, using 'foo' as a parameter.

\A'*ident*' [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
```

[15] Consider what happens when a C1 control 0x80–0x9F is necessary as a continuation byte in a UTF-8 sequence.

See Section 5.5.3 [Escapes], page 73, 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 191., Section 5.6.2 [Interpolating Registers], page 79, and Section 5.19 [Strings], page 137.

Macros, strings, and diversions (and boxes) share the same name space.

```
.de xxx
.  nop foo
..
.
.di xxx
bar
.br
.di
.
.xxx
    ⇒ bar
```

As the previous example shows, GNU `troff` reuses the identifier '`xxx`', changing it from a macro to a diversion. No warning is emitted! The contents of the first macro definition are lost.

See Section 5.6.2 [Interpolating Registers], page 79, and Section 5.19 [Strings], page 137.

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* that takes up an entire line, and does some large-scale operation (e.g. break lines, start new pages).

The other is an *escape* that can be usually embedded anywhere in the text; most requests can accept it even as an argument. Escapes generally do more minor operations like sub- and superscripts, print a symbol, etc.

5.5.1 Requests

A request line begins with a control character, which is either a single quote (‘'’, the *no-break control character*) or a period (‘.’, the normal *control character*). These can be changed; see Section 5.11 [Character Translations], page 101, 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 spaces and tabs are ignored after a control character, it is common practice to use them to structure the source of documents or macro files.

```
.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 167.

To begin a line with a control character without it being interpreted, precede it with `\&`. This represents a non-printing input break, which means it does not affect the output.

In most cases the period is used as a control character. Several requests cause a break implicitly; using the single quote control character prevents this.

`\n[.br]` [Register]

A read-only number register, which is set to 1 if a macro is called with the normal control character (as defined with the `cc` request), and set to 0 otherwise.

This allows reliable modification of requests.

```
.als bp*orig bp
.de bp
.  tm before bp
.  ie \\n[.br] .bp*orig
.  el 'bp*orig
.  tm after bp
..
```

Using this register outside of a macro makes no sense (it always returns zero in such cases).

If a macro is called as a string (that is, using `\*`), the value of the `.br` register is inherited from the caller.

5.5.1.1 Request and Macro Arguments

Arguments to requests and macros are separated by space characters.[16] Only one space between arguments is necessary; additional ones are harmless and ignored.

A macro argument that must contain space characters can either be enclosed in double quotes—this is *not* true of requests—or one of several varieties of *escape* with a spacing function can be used instead.

Consider calls to a hypothetical macro `uh`:

```
.uh The Mouse Problem
.uh "The Mouse Problem"
.uh The\~Mouse\~Problem
.uh The\ Mouse\ Problem
```

The first line is the `uh` macro being called with three arguments, '`The`', '`Mouse`', and '`Problem`'. The remainder call the `uh` macro with one argument, '`The Mouse Problem`'. The last solution, using escaped spaces, is "classical" in the sense that it can be found in documents prepared for AT&T `troff`. Nevertheless, it is not optimal in most situations, since '`\ `' inserts a fixed-width, non-breaking space character that can't be adjusted. GNU `troff` provides a different command `\~` to insert a adjustable, non-breaking space.[17]

A double quote that isn't preceded by a space doesn't start a macro argument. If not closing a string, it is printed literally.

---

[16] Plan 9 `troff` also allows tabs for argument separation—GNU `troff` intentionally doesn't support this.

[17] `\~` is also supported by Heirloom Doctools `troff` 050915 (September 2005) and mandoc 1.14.5 (March 2019) but not by Plan 9 `troff`, Solaris `troff`, DWB `troff` or `onroff`, or `neatroff`.

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 GNU `troff` doesn’t convert `\(dq` back to a double-quote input character.

  This method won’t work with AT&T `troff` since it doesn’t define the ‘dq‘ special character.

Double quotes in the `ds` request are handled differently. See Section 5.19 [Strings], page 137, for more details.

5.5.2 Macros

`gtroff` has a *macro* facility for defining a series of lines that can be invoked by name. They are called in the same manner as requests—arguments also may be passed basically in the same manner.

See Section 5.21 [Writing Macros], page 148, and Section 5.5.1.1 [Request and Macro Arguments], page 72.

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 101.

Escape sequences that require an identifier as a parameter accept three possible syntax forms.

- The next single character is the identifier.
- If this single character is an opening parenthesis, take the following two characters as the identifier. 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'
```

The quote character can be replaced with any other character that does not occur in the argument (even a newline or a space character) in the following escapes: `\o`, `\b`, and `\X`. This makes e.g.

```
A caf
\o
e\'


in Paris
  ⇒ A café in Paris
```

possible, but it is better not to use this feature to avoid confusion.

The following escape 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 101, for an exact description of those escapes.

See Section 5.34 [Implementation Differences], page 193, Section 5.21.1 [Copy Mode], page 151, Section 5.25 [Diversions], page 170, and Section 5.4 [Identifiers], page 69.

5.5.3.1 Comments

Probably one of the most[18] 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 idiosyncrasy is that tabs must not be used to line up comments. Tabs are not treated as whitespace between the request and macro arguments.

A comment on a line by itself is treated as a blank line, because after eliminating the comment, that is all that remains:

```
Test
\" comment
Test
```

produces

```
Test

Test
```

To avoid this, it is common to start the line with .\", which causes the line to be treated as an undefined request and thus ignored completely.

Another commenting scheme seen sometimes is three consecutive single quotes (''') at the beginning of a line. This works, but gtroff gives a warning about an undefined macro (namely ''), which is harmless, but irritating.

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

`\#` [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...
```

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 79).

5.6 Registers

Numeric variables in GNU `troff` 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 69, for details on register identifiers.

5.6.1 Setting Registers

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

Although the following requests and escapes can be used to create registers, simply using an undefined register will cause it to be set to zero.

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

Set number register *ident* to *value*. If *ident* doesn't exist, GNU `troff` creates it.

The argument to `\R` usually has to be enclosed in quotes. See Section 5.5.3 [Escapes], page 73, for details on parameter delimiting characters.

(Later, we will discuss additional forms of `nr` and `\R` that can change a register's value after it is dereferenced. Section 5.6.3 [Auto-increment], page 79.)

The `\R` escape doesn't produce an input token in GNU `troff`; in other words, it vanishes completely after GNU `troff` has processed it.

For example, the following two lines are equivalent:

```
.nr a (((17 + (3 * 4))) % 4)
\R'a (((17 + (3 * 4))) % 4)'
    ⇒ 1
```

The complete transparency of `\R` can cause surprising effects if you use number registers like `.k`, which get evaluated at the time they are accessed.

```
.ll 1.6i
.
aaa bbb ccc ddd eee fff ggg hhh\R':k \n[.k]'
.tm :k == \n[:k]
    ⇒ :k == 126950
.
.br
.
aaa bbb ccc ddd eee fff ggg hhh\h'0'\R':k \n[.k]'
.tm :k == \n[:k]
    ⇒ :k == 15000
```

If you process this with the PostScript device (`-Tps`), there will be a line break eventually after `ggg` in both input lines. However, after processing the space after `ggg`, the partially collected line is not overfull yet, so GNU `troff` continues to collect input until it sees the space (or in this case, the newline) after `hhh`. At this point, the line is longer than the line length, and the line gets broken.

In the first input line, since the `\R` escape leaves no traces, the check for the overfull line hasn't been done yet at the point where `\R` gets handled, and you get a value for the `.k` number register that is even greater than the current line length.

In the second input line, the insertion of `\h'0'` to emit an invisible zero-width space forces GNU `troff` to check the line length, which in turn causes the start of a new output line. Now `.k` returns the expected value.

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

`.nr` *ident* `+`*value* [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. Technically, only the name is removed; the register's contents are still accessible under aliases created with `aln`, if any.

`.rnn` *ident1 ident2* [Request]

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

`.aln` *new old* [Request]

Create an alias *new* for an existing number register *old*, causing the names to refer to the same stored object. If *old* is undefined, a warning of type '`reg`' is generated and the request is ignored. See Section 5.33 [Debugging], page 188, for information about warnings.

To remove a number register alias, call `rr` on its name. A number register's contents do not become inaccessible until it has no more names.

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*. 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+[`*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, it is always textually replaced (or interpolated) with a representation of that number. This output format can be changed to a variety of formats (numbers, Roman numerals, etc.). This is done using the `af` request.

`.af` *ident format* [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 does work; `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 188, 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 writable register, then apply the `af` request to this other register.

`\g`*i* [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 that are not described elsewhere in this manual. Any register that begins with a '`.`' is read-only. A complete listing of all built-in registers can be found in tie E [Register Index], page 249.

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

`\n[.H]` Number of basic units per horizontal unit of output device resolution. See Section 5.2 [Measurements], page 66.

`\n[.R]` The number of number registers available. This is always 10000 in GNU `troff`; it exists for backward compatibility.

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

`\n[.V]` Number of basic units per vertical unit of output device resolution. See Section 5.2 [Measurements], page 66.

`\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 184, 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[.O]` This read-only register is set to the suppression nesting level (see escapes `\O`). See Section 5.27 [Suppressing output], page 176.

`\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.

5.7 Manipulating Filling and Adjustment

Various ways of causing *breaks* were given in Section 5.1.4 [Breaking], 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 174).

See Section 5.14 [Line Control], page 109, 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 request 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 174).

See Section 5.14 [Line Control], page 109, 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.

Using `ad` without argument is the same as saying '`.ad \n[.j]`'. In particular, `gtroff` adjusts lines in the same way it did before adjusting was deactivated (with a call to `na`, say). For example, this input code

```
.de AD
.  br
.  ad \\$1
..
.
.de NA
.  br
.  na
..
.
textA
.AD r
.nr ad \n[.j]
textB
.AD c
textC
.NA
textD
.AD         \" back to centering
textE
.AD \n[ad]  \" back to right justifying
textF
```

produces the following output:

```
textA
                                                              textB
                              textC
textD
                              textE
                                                              textF
```

As just demonstrated, the current adjustment mode is available in the read-only number register `.j`; it can be stored and subsequently used to set adjustment.

The adjustment mode status is associated with the current environment (see Section 5.26 [Environments], page 174).

`.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 174).

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

Break, adjusting the current line per the current adjustment mode.

With `\p`, this break will happen at the next word boundary. The `\p` itself is removed entirely, adding neither a break nor a space where it appears

in input; it can thus be placed in the middle of a word to cause a break at the end of that word.

In most cases this produces very ugly results since `gtroff` doesn't have a sophisticated paragraph building algorithm (as TEX has, 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]

Set the sizes of spaces between words and sentences. Their units are twelfths of the space width parameter of the current font. Initially both the *word-space-size* and *sentence-space-size* are 12. Negative values are not permitted. The request is ignored if there are no arguments.

The first argument, the inter-word space size, is a minimum; if automatically adjusted, it may increase.

The optional second argument sets the amount of additional space separating sentences on the same output line in fill mode. If the second argument is omitted, *sentence-space-size* is set to *word-space-size*.

The read-only number registers `.ss` and `.sss` hold the values of minimal inter-word space and additional inter-sentence space, respectively. These parameters are associated with the current environment (see Section 5.26 [Environments], page 174), and rounded down to the nearest multiple of 12 on terminal output devices.

Additional inter-sentence spacing is used only in fill mode, and only if the output line is not full when the end of a sentence occurs in the input. If a sentence ends at the end of an input line, then both an inter-word space and an inter-sentence space are added to the output; if two spaces follow the end of a sentence in the middle of an input line, then the second space becomes an inter-sentence space in the output. Additional inter-sentence space is not adjusted, but the inter-word space that always precedes it may be. Further input spaces after the second, if present, are adjusted as normal.

If a second argument is never given to the `ss` request, GNU `troff` separates sentences as AT&T `troff` does. In input to GNU `troff`, as with AT&T `troff`, a sentence should always be followed by either a newline or two spaces.

A related application of the `ss` request is to insert discardable horizontal space; i.e., space that is discarded at a line break. For example, some

footnote styles collect the notes into a single paragraph with large spaces between each.

```
.ie n .ll 50n
.el   .ll 2.75i
.ss 12 48
1. J. Fict. Ch. Soc. 6 (2020), 3\[en]14.
2. Better known for other work.
```

The result has obvious inter-sentence spacing.

```
1. J. Fict. Ch. Soc. 6 (2020), 3-14.     2. Better
known for other work.
```

If *undiscardable* space is required, use the `\h` escape.

`.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 174).

The following example demonstrates the differences.

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

.ad c
This is a small text fragment that shows the differences
between the ‘.ce’ and the ‘.ad c’ request.
    ⇒   This is a small text fragment that
    ⇒            shows the differences
    ⇒ between the ‘.ce’ and the ‘.ad c’ request.
    ⇒
    ⇒   This is a small text fragment that
    ⇒ shows the differences between the ‘.ce’
    ⇒          and the ‘.ad c’ request.
```

With no arguments, `ce` centers the next line of text. *nnn* specifies the number of lines to be centered. If the argument is zero or negative, centering is disabled.

The basic length for centering text is the line length (as set with the `ll` request) minus the indentation (as set with the `in` request). Temporary indentation is ignored.

The previous example shows the common idiom of turning on centering for a large number of lines, and turning off centering after the text to be centered. This is useful for any request that takes a number of lines as an argument.

The `.ce` read-only number register contains the number of lines remaining to be centered, as set by the `ce` request.

`.rj` [*nnn*] [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 174).

5.8 Manipulating Hyphenation

GNU `troff` hyphenates words automatically by default. Automatic hyphenation of words in natural languages is a subject requiring algorithms and data, and is susceptible to conventions and preferences. Before tackling automatic hyphenation, let us consider how it can be done manually.

Explicitly hyphenated words such as "mother-in-law" are eligible for breaking after each of their hyphens when GNU `troff` fills lines. Relatively few words in a language offer such obvious break points, however, and automatic hyphenation is not perfect, particularly for unusual words found in domain-specific jargon. We may wish to explicitly instruct GNU `troff` how to hyphenate words if the need arises.

`.hw` *word* ... [Request]

Define each *hyphenation exception word* with each hyphen '-' in the word indicating a hyphenation point. For example, the request

```
.hw in-sa-lub-rious alpha
```

marks potential hyphenation points in "insalubrious", and prevents "alpha" from being hyphenated at all.

Besides the space character, any character whose hyphenation code is zero can be used to separate the arguments of `hw` (see the `hcode` request below). In addition, this request can be used more than once.

Hyphenation points specified with `hw` are not subject to the restrictions given by the `hy` request (see below).

Hyphenation exceptions specified with the `hw` request are associated with the hyphenation language (see below) and environment (see Section 5.26 [Environments], page 174); calling the `hw` request in the absence of a hyphenation language is an error.

The request is ignored if there are no parameters.

These are known as hyphenation *exceptions* in the expectation that most users will avail themselves of automatic hyphenation; these exceptions override any rules that would normally apply to a word matching a hyphenation exception defined with `hw`.

Situations also arise when only a specific occurrence of a word needs its hyphenation altered or suppressed, or when something that is not a word in a natural language, like a URL, needs to be broken in sensible places without hyphens.

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

To tell GNU `troff` how to hyphenate words as they occur in input, use the `\%` escape, also known as the *hyphenation character*. Preceding a word with this escape prevents it from being automatically hyphenated; each instance within a word indicates to GNU `troff` that the word may be hyphenated at that point. This mechanism affects only that occurrence of the word; to change the hyphenation of a word for the remainder of the document, use the `hw` request.

GNU `troff` regards the escapes `\X` and `\Y` as starting a word; that is, the `\%` escape in, say, '`\X'...'\%foobar`' or '`\Y'...'\%foobar`' no longer prevents hyphenation of '`foobar`' but inserts a hyphenation point just prior to it; most likely this isn't what you want. See Section 5.30 [Postprocessor Access], page 183.

The `\:` escape inserts a non-printing break point; that is, the word can break there, but the soft hyphen glyph is not written to the output if it does. Breaks are word boundaries, so if a break is inserted, the remainder of the (input) word is subject to hyphenation as normal.

You can use `\:` and `\%` in combination to control breaking of a file name or URL.

```
... check \%/var/log/\:\%httpd/\:\%access_log ...
```

`.hc` [*char*] [Request]

Change the hyphenation character to *char*. This character then works as the `\%` escape normally does, and thus no longer appears in the output.[19] Without an argument, `hc` resets the hyphenation character to `\%` (the default).

The hyphenation character is associated with the current environment (see Section 5.26 [Environments], page 174).

`.shc` [*glyph*] [Request]

Set the *soft hyphen character* to *glyph*.[20] If the argument is omitted, the soft hyphen character is set to the default, `\[hy]`. The *soft hyphen character* is the glyph that is inserted when a word is automatically hyphenated at a line break.[21] If the soft hyphen character does not exist in the font of the character immediately preceding a potential break point,

[19] `\%` itself stops marking hyphenation points but still produces no output glyph.

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

[21] It is "soft" because it only appears in output where hyphenation is actually performed; a "hard" hyphen, as in "long-term", always appears.

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 assigning the soft hyphen character.

Several requests influence automatic hyphenation. Because conventions vary, a variety of hyphenation modes are available to the `hy` request; these determine whether automatic hyphenation will apply to a word prior to breaking a line at the end of a page (more or less; see below for details), and at which positions within that word hyphenation is permissible. The places within a word that are eligible for hyphenation are determined by language-specific data and lettercase relationships. Furthermore, hyphenation of a word might be suppressed because too many previous lines have been hyphenated (`hlm`), the line has not reached a certain minimum length (`hym`), or the line can instead be adjusted with up to a certain amount of additional inter-word space (`hys`).

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

Set hyphenation mode to *mode*. The optional numeric argument *mode* encodes conditions for hyphenation.

Typesetting practice generally does not avail itself of every opportunity for hyphenation, but the details differ by language and site mandates. The hyphenation modes of AT&T `troff` were implemented with English-language publishing practices of the 1970s in mind, not a scrupulous enumeration of conceivable parameters. GNU `troff` extends those modes such that finer-grained control is possible, retaining compatibility with older implementations at the expense of a more intuitive arrangement. The means of hyphenation mode control is a set of numbers that can be added up to encode the behavior sought.[22] The entries in the table below are termed *values*, and the sum of the desired values is the *mode*.

`0` disables hyphenation.

`1` enables hyphenation except after the first and before the last character of a word; this is the default if *mode* is omitted and also the start-up value of GNU `troff`.

The remaining values "imply" 1; that is, they enable hyphenation under the same conditions as '`.hy 1`', and then apply or lift restrictions relative to that basis.

`2` disables hyphenation of the last word on a page.[23]

[22] The mode is a vector of booleans encoded as an integer. To a programmer, this fact is easily deduced from the exclusive use of powers of two for the configuration parameters; they are computationally easy to "mask off" and compare to zero. To almost everyone else, the arrangement seems recondite and unfriendly.

[23] This value prevents hyphenation if the next page location trap is closer than the next text baseline would be. GNU `troff` automatically inserts an implicit vertical position

4 disables hyphenation before the last two characters of a word.

8 disables hyphenation after the first two characters of a word.

16 enables hyphenation before the last character of a word.

32 enables hyphenation after the first character of a word.

Any restrictions imposed by the hyphenation mode are *not* respected for words whose hyphenations have been explicitly specified with the hyphenation character ('\%' by default) or the `hw` request.

The nonzero values in the previous table are additive. For example, value 12 causes GNU `troff` to hyphenate neither the last two nor the first two characters of a word. Some values cannot be used together because they contradict; for instance, values 4 and 16, and values 8 and 32. As noted, it is superfluous to add 1 to any other positive value.

The automatic placement of hyphens in words is determined by *pattern files*, which are derived from TEX and available for several languages. The number of characters at the beginning of a word after which the first hyphenation point should be inserted is determined by the patterns themselves; it can't be reduced further without introducing additional, invalid hyphenation points (unfortunately, this information is not part of a pattern file—you have to know it in advance). The same is true for the number of characters at the end of a word before the last hyphenation point should be inserted. For example, you can supply the following input to '`echo $(nroff)`'.

```
.ll 1
.hy 48
splitting
```

You will get

```
s- plit- t- in- g
```

instead of the correct 'split- ting'. U.S. English patterns as distributed with GNU `troff` need two characters at the beginning and three characters at the end; this means that value 4 of `hy` is mandatory. Value 8 is possible as an additional restriction, but values 16 and 32 should be avoided, as should mode 1 (the default!). Modes 4 and 6 are typical.

A table of left and right minimum character counts for hyphenation as needed by the patterns distributed with GNU `troff` follows; see the *groff_tmac(5)* man page (type `man groff_tmac` at the command line) for more information on GNU `troff`'s language macro files.

language	**pattern name**	**left min**	**right min**
Czech	cs	2	2

trap at the end of each page to cause a page transition. This value can be used in traps planted by users or macro packages to prevent hyphenation of the last word in a column in multi-column page layouts or before floating figures or tables. See Section 5.24.1 [Page Location Traps], page 163.

U.S. English	us	2	3
French	fr	2	3
German traditional	det	2	2
German reformed	den	2	2
Swedish	sv	1	2

Hyphenation exceptions within pattern files (i.e., the words within a TEX `\hyphenation` group) also obey the hyphenation restrictions given by `hy`. However, exceptions specified with `hw` do not.

The hyphenation mode is associated with the current environment (see Section 5.26 [Environments], page 174).

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

`.nh` [Request]

Disable hyphenation; i.e., set the hyphenation mode to 0 (see above). The hyphenation mode of the last call to `hy` is not remembered.

`.hpf` *pattern-file* [Request]
`.hpfa` *pattern-file* [Request]
`.hpfcode` *a b* [*c d*] ... [Request]

Read hyphenation patterns from *pattern-file*. This file is sought in the same way that macro files are with the `mso` request or the `-mname` command-line option to `groff`.

The *pattern-file* should have the same format as (simple) TEX pattern 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.
- "Digraphs" like `\$` are not supported.
- `^^xx` (where each *x* is 0–9 or a–f) and `^^c` (character *c* in the code point range 0–127 decimal) are recognized; other uses of `^` cause an error.
- No macro expansion is performed.
- `hpf` checks for the expression `\patterns{...}` (possibly with white-space before or after the braces). Everything between the braces is taken as hyphenation patterns. Consequently, `{` and `}` are not allowed in patterns.
- Similarly, `\hyphenation{...}` gives a list of hyphenation exceptions.
- `\endinput` is recognized also.
- For backwards compatibility, if `\patterns` is missing, the whole file is treated as a list of hyphenation patterns (except that the `%` character is recognized as the start of a comment).

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

The `hpfcode` request defines mapping values for character codes in pattern files. It is an older mechanism no longer used by GNU `troff`'s own

macro files; for its successor, see hcode below. hpf or hpfa apply the mapping after reading the patterns but before replacing or appending to the active list of patterns. Its arguments are pairs of character codes—integers from 0 to 255. The request maps character code *a* to code *b*, code *c* to code *d*, and so on. Character codes that would otherwise be invalid in GNU troff can be used. By default, every code maps to itself except those for letters 'A' to 'Z', which map to those for 'a' to 'z'.

The set of hyphenation patterns is associated with the 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 U.S. English (in files hyphen.us and hyphenex.us).

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

Invoking hpf or hpfa causes an error if there is no hyphenation language.

If no hpf request is specified (either in the document, in a troffrc or troffrc-end file, or in a macro package), GNU troff won't automatically hyphenate at all.

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

Set the hyphenation code of character *c1* to *code1*, that of *c2* to *code2*, and so on. A hyphenation code must be a single input character (not a special character) other than a digit or a space. The request is ignored if it has no parameters.

For hyphenation to work, hyphenation codes must be set up. At startup, GNU troff assigns hyphenation codes to the letters 'a'–'z' (mapped to themselves), to the letters 'A'–'Z' (mapped to 'a'–'z'), and zero to all other characters. Normally, hyphenation patterns contain only lowercase letters which should be applied regardless of case. In other words, they assume that the words 'FOO' and 'Foo' should be hyphenated exactly as 'foo' is. The hcode request extends this principle to letters outside the Unicode basic Latin alphabet; without it, words containing such letters won't be hyphenated properly even if the corresponding hyphenation patterns contain them. For example, the following hcode requests are necessary to assign hyphenation codes to the letters 'ÄäÖöÜüß' (needed for German):

```
.hcode ä ä  Ä ä
.hcode ö ö  Ö ö
.hcode ü ü  Ü ü
.hcode ß ß
```

Without those assignments, GNU troff treats German words like 'Kindergärten' (the plural form of 'kindergarten') as two substrings 'kinderg' and 'rten' because the hyphenation code of the umlaut a is zero by default. There is a German hyphenation pattern that covers 'kinder', so GNU troff finds the hyphenation 'kin-der'. The other two hyphenation points ('kin-der-gär-ten') are missed.

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

Set the hyphenation language to *lang*. Hyphenation exceptions specified with the `hw` request and hyphenation patterns and exceptions specified with the `hpf` and `hpfa` requests are associated with the hyphenation language. The `hla` request is usually invoked by the `troffrc` or `troffrc-end` files; `troffrc` sets the default language to '`us`' (U.S. English).

The hyphenation language is associated with the current environment (see Section 5.26 [Environments], page 174).

The hyphenation language is available as a string in the read-only number register '`.hla`'.

```
.ds curr_language \n[.hla]
\*[curr_language]
    ⇒ us
```

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

Set the maximum number of consecutive hyphenated lines to *n*. If *n* is negative, there is no maximum. If omitted, *n* is −1. This value is associated with the current environment (see Section 5.26 [Environments], page 174). 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 `.hlm` read-only number register stores this maximum. The count of immediately preceding consecutive hyphenated lines is available in the read-only number register `.hlc`.

`.hym` [*length*] [Request]
`\n[.hym]` [Register]

Set the (right) hyphenation margin to *length*. If the adjustment mode is not '`b`' or '`n`', the line is not hyphenated if it is shorter than *length*. Without an argument, the hyphenation margin is reset to its default value, 0. The default scaling indicator is '`m`'. The hyphenation margin is associated with the current environment (see Section 5.26 [Environments], page 174).

A negative argument resets the hyphenation margin to zero, emitting a warning of type '`range`'.

The hyphenation margin is available in the `.hym` read-only number register.

`.hys` [*hyphenation-space*] [Request]
`\n[.hys]` [Register]

Suppress hyphenation of the line in adjustment modes '`b`' or '`n`' if it can be justified by adding no more than *hyphenation-space* extra space to

each inter-word space. Without an argument, the hyphenation space adjustment threshold is set to its default value, 0. The default scaling indicator is 'm'. The hyphenation space adjustment threshold is associated with the current environment (see Section 5.26 [Environments], page 174).

A negative argument resets the hyphenation space adjustment threshold to zero, emitting a warning of type 'range'.

The hyphenation space adjustment threshold is available in the .hys read-only number register.

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, and that adds the current line spacing to the space you have just specified. The default scaling indicator is 'v'.

For convenience you may wish to use the following macros to set the height of the next line at a given distance from the top or the bottom of the page:

```
.de y-from-top-down
.  sp |\\$1-\\n[.v]u
..
.
.de y-from-bot-up
.  sp |\\n[.p]u-\\$1-\\n[.v]u
..
```

A call to '.y-from-bot-up 10c' means that the bottom of the next line will be at 10 cm from the paper edge at the bottom.

If a vertical trap is sprung during execution of sp, the amount of vertical space after the trap is discarded. For example, this

```
.de xxx
..
.
.wh 0 xxx
.
.pl 5v
foo
.sp 2
bar
.sp 50
baz
```

results in

```
foo


bar


baz
```

The amount of discarded space is available in the number register `.trunc`.

To protect `sp` against vertical traps, use the `vpt` request:

```
.vpt 0
.sp -3
.vpt 1
```

`.ls` [*nnn*] [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 174).

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

See Section 5.18.1 [Changing Type Sizes], page 133, 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 73, for details on parameter delimiting characters.

The `.a` read-only number register contains the most recent (non-negative) 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 112). 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 Mode], page 151), `\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`') that indicate where each tab stop is to be (overriding any previous settings).

Tab stops can be specified absolutely, i.e., as the distance from the left margin. For example, the following sets 6 tab stops every one inch.

```
.ta 1i 2i 3i 4i 5i 6i
```

Tab stops can also be specified using a leading '`+`', which means that the specified tab stop is set relative to the previous tab stop. For example, the following is equivalent to the previous example.

```
.ta 1i +1i +1i +1i +1i +1i
```

`gtroff` supports an extended syntax to specify repeat values after the '`T`' mark (these values are always taken as relative)—this is the usual way to specify tabs set at equal intervals. The following is, yet again, the same as the previous examples. It does even more since it defines an infinite number of tab stops separated by one inch.

```
.ta T 1i
```

Now we are ready to interpret the full syntax given at the beginning: Set tabs at positions *n1*, *n2*, ..., *nn* and then set tabs at *nn*+*r1*, *nn*+*r2*, ..., *nn*+*rn* and then at *nn*+*rn*+*r1*, *nn*+*rn*+*r2*, ..., *nn*+*rn*+*rn*, and so on.

Example: '`4c +6c T 3c 5c 2c`' is equivalent to '`4c 10c 13c 18c 20c 23c 28c 30c ...`'.

The material in each tab column (i.e., the column between two tab stops) may be justified to the right or left or centered in the column. This is specified by appending '`R`', '`L`', or '`C`' to the tab specifier. The default justification is '`L`'. Example:

```
.ta 1i 2iC 3iR
```

Some notes:

- The default unit of the `ta` request is '`m`'.
- A tab stop is converted into a non-breakable horizontal movement that can be neither stretched nor squeezed. For example,

  ```
  .ds foo a\tb\tc
  .ta T 5i
  \*[foo]
  ```

  creates a single line, which is a bit longer than 10 inches (a string is used to show exactly where the tab characters are). Now consider the following:

  ```
  .ds bar a\tb b\tc
  .ta T 5i
  \*[bar]
  ```

  `gtroff` first converts the tab stops of the line into unbreakable horizontal movements, then splits the line after the second '`b`' (assuming a sufficiently short line length). Usually, this isn't what the user wants.
- Superfluous tabs (i.e., tab characters that do not correspond to a tab stop) are ignored except the first one, which delimits the char-

acters 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 174).
- Calling `ta` without an argument removes all tab stops.
- The start-up value of `gtroff` is '`T 0.5i`'.

The read-only number register `.tabs` contains a string representation of the current tab settings suitable for use as an argument to the `ta` request.

```
.ds tab-string \n[.tabs]
\*[tab-string]
    ⇒ T120u
```

The `troff` version of the Plan 9 operating system uses register `.S` for the same purpose.

`.tc` [*fill-glyph*] [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 174).[24]

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

`.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:

```
.ds x a\t\c
.ds y b\t\c
.ds z c
.ta 1i 3i
\*x
\*y
\*z
```

in normal mode, results in the output

```
a       b       c
```

in line-tabs mode, the same code outputs

```
a       b                       c
```

Line-tabs mode is associated with the current environment. The read-only register `.linetabs` is set to 1 if in line-tabs mode, and 0 in normal mode.

5.10.1 Leaders

Sometimes it may be desirable to use the `tc` request to fill a particular tab stop with a given glyph (for example dots in a table of contents), but also normal tab stops on the rest of the line. For this GNU `troff` provides an alternate tab mechanism, called *leaders*, which does just that.[25]

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 Mode], page 151), `\a` is the same as a real leader character.

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

Declare the *leader repetition character*.[26] 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 174).

[25] This is pronounced to rhyme with “feeder”, and refers to how the glyphs “lead” the eye across the page to the corresponding page number or other datum.

[26] *Leader repetition character* is a misnomer since it is an output glyph.

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]
```

This produces

```
1.1  Foo.....................................  12
```

5.10.2 Fields

Fields are a more general way of laying out tabular data. A field is defined as the data between a pair of *delimiting characters*. It contains substrings that are separated by *padding characters*. The width of a field is the distance on the *input* line from the position where the field starts to the next tab stop. A padding character inserts stretchable space similar to 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). In contrast to, e.g., the tab repetition character, delimiting and padding characters are *not* associated with the current environment (see Section 5.26 [Environments], page 174).

```
.fc # ^
.ta T 3i
#foo^bar^smurf#
.br
#foo^^bar^smurf#
    ⇒ 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 174).

`.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 174).

See Section 5.5.1 [Requests], page 71.

`.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.

Changing the escape character globally likely breaks macro packages, since GNU `troff` has no mechanism to 'intern' macros, i.e., to convert a macro definition into an internal form that is independent of its representation (TEX has such a 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 the top level or in a diversion is a backslash glyph printed; in copy mode, it expands to a single backslash, which then combines with the following character to form an escape sequence.

The \E escape differs from \e by printing an escape character that is not interpreted in copy mode. Use this to define strings with escapes that work when used in copy mode (for example, as a macro argument). The following example defines strings to begin and end a superscript:

```
.ds { \v'-.3m'\s'\En[.s]*60/100'
.ds } \s0\v'.3m'
```

Another example to demonstrate the differences between the various escape sequences, using a strange escape character, '-'.

```
.ec -
.de xxx
--A'foo'
..
.xxx
    ⇒ -A'foo'
```

The result is surprising for most users, expecting '1' since 'foo' is a valid identifier. What has happened? As mentioned above, the leading escape character makes the following character ordinary. Written with the default escape character the sequence '--' becomes '\-'—this is the minus sign.

If the escape character followed by itself is a valid escape sequence, only \E yields the expected result:

```
.ec -
.de xxx
-EA'foo'
..
.xxx
    ⇒ 1
```

`\.` [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 executing macro `foo`.

A *translation* is a mapping of an input character to an output glyph. The mapping occurs at output time, i.e., the input character gets assigned the metric information of the mapped output character right before input tokens are converted to nodes (see Section 5.32 [Gtroff Internals], page 186, 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 170, 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 non-printing input break) 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 186.

- `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 that shows up in warnings like '`can't find character with input code` *`XXX`*' if the `tr` request isn't handled properly.

  Consider the following translation:

  ```
  .tr éÉ
  ```

  This maps input character `é` onto glyph `É`, which is identical to glyph `char201`. But this glyph intentionally doesn't exist! Instead, `\[char201]` is treated as an input character entity and is by default mapped onto `\['E]`, and `gtroff` doesn't handle translations of translations.

  The right way to write the above translation is

  ```
  .tr é\['E]
  ```

  In other words, the first argument of `tr` should be an input character or entity, and the second one a glyph entity.

- Without an argument, the `tr` request is ignored.

`.trnt` *abcd...* [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 170.

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`.

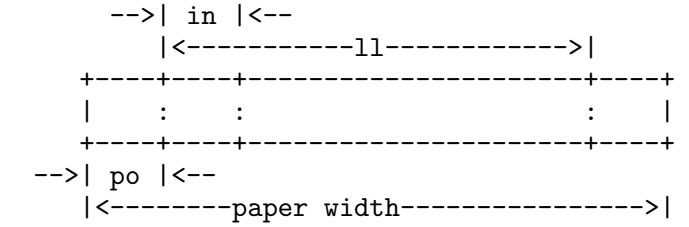
`.troff` [Request]
Make the '`t`' built-in condition true (and the '`n`' built-in condition false) for `if`, `ie`, and `while` conditional requests. This is the default if `gtroff` (*not* `groff`) is started with the `-R` switch to avoid loading of the start-up files `troffrc` and `troffrc-end`. Without `-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 143, for more details on built-in conditions.

5.13 Line Layout

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



These dimensions are:

`po` *Page offset*—this is the leftmost position of text on the final output, defining the *left margin*.

`in` *Indentation*—this is the distance from the left margin where text is printed.

`ll` *Line length*—this is the distance from the left margin to right margin.

A simple demonstration:

```
.ll 3i
This is text without indentation.
The line length has been set to 3\~inch.
.in +.5i
.ll -.5i
Now the left and right margins are both increased.
.in
.ll
Calling .in and .ll without parameters restore
the previous values.
```

Result:

```
This  is text without indenta-
tion.   The  line  length  has
been set to 3 inch.
     Now   the  left  and
     right  margins   are
     both increased.
Calling  .in  and  .ll without
parameters restore the  previ-
ous values.
```

`.po` [*offset*] [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*). 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 terminal 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 AT&T `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 174).

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 174). 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 6.5i. The effect of ll is delayed until a partially collected line (if it exists) is output. The default scaling indicator is 'm'.

If ll is called without an argument, the line length is reset to the previous value before the last call to ll. If a negative line length is specified (which is not allowed), gtroff emits a warning of type 'range' and sets the line length to zero.

The line length is associated with the current environment (see Section 5.26 [Environments], page 174).

The current line length (as set by ll) can be found in the read-only number register '.l'. The read-only number register .ll is the line length that applies to the current output line.

Similar to .i and .in, the difference between .l and .ll is that the latter takes into account whether a partially collected line still uses the old line length value.

5.14 Line Control

It is important to understand how gtroff handles input and output lines.

Many escapes use positioning relative to the input line. For example, this

```
This is a \h'|1.2i'test.

This is a
\h'|1.2i'test.
```

produces

```
This is a   test.

This is a             test.
```

The main usage of this feature is to define macros that act exactly at the place where called.

```
.\" A simple macro to underline a word
.de underline
.  nop \\$1\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 is appended to the current partial output line. The next non-command line after an interrupted line counts as a new input line.

The visual results depend on whether no-fill mode is active.

- If no-fill mode is active (using the `nf` request), the next input text line after `\c` is handled as a continuation of the same input text line.

```
.nf
This is a \c
test.
    ⇒ This is a test.
```

- If fill mode is active (using the `fi` request), a word interrupted with `\c` is continued with the text on the next input text line, without an intervening space.

```
This is a te\c
st.
    ⇒ This is a test.
```

An intervening control line that causes a break is stronger than `\c`, flushing out the current partial line in the usual way.

The `.int` register contains a positive value if the last output line was interrupted with `\c`; this is associated with the current environment (see Section 5.26 [Environments], page 174).

5.15 Page Layout

GNU `troff` provides some 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`'.

This specifies only the size of the page, not the top and bottom margins. Those are not set by GNU `troff` directly. See Section 5.24 [Traps], page 163, 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.

GNU `troff` provides several operations that help in setting up top and bottom titles (also known as 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:

- The line length set by the `ll` request is not honoured by `tl`; use the `lt` request (described below) instead, to control line length for text set by `tl`.
- 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-justified parts are empty.
- `tl` accepts the same parameter delimiting characters as the `\A` escape; see Section 5.5.3 [Escapes], page 73.

`.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 6.5i. If a negative line length is specified (which is not allowed), `gtroff` emits a warning of type '`range`' and sets the title line length to zero. The default scaling indicator is '`m`'. If `lt` is called without an argument, the title length is reset to the previous value before the last call to `lt`.

The current setting of this is available in the `.lt` read-only number register; it is associated with the current environment (see Section 5.26 [Environments], page 174).

`.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. This doesn't affect the number register `%`.

See Section 5.24 [Traps], page 163.

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 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 111.

```
.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 170).

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 163.

.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 163, for further information). The default scaling indicator for ne is 'v'; the default value of *space* is 1 v if no argument is given.

For example, to make sure that no fewer than 2 lines get orphaned, do the following before each paragraph:

```
.ne 2
text text text
```

ne then automatically causes a page break if there is space for one line only.

.sv [*space*] [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 ignores 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 170, 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-terminal devices, there is also at least one symbol font that 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, the `\F` escape, or the `family` command in the `DESC` file).

It is not possible to switch to a font with the name '`DESC`' (whereas this name could be used as a style name; however, this is not recommended).

With no argument or using '`P`' as an argument, `ft` switches to the previous font. Use `\f[]` to do this with the escape. The old syntax forms `\fP` or `\f[P]` are also supported.

Fonts are generally specified as upper-case strings, which are usually 1 to 4 characters representing an abbreviation or acronym of the font name. This is no limitation, just a convention.

The example below produces two identical lines.

```
eggs, bacon,
.ft B
spam
.ft
and sausage.

eggs, bacon, \fBspam\fP and sausage.
```

`\f` doesn't produce an input token in GNU `troff`. 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 118, 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.

Font translations cannot be chained.

```
.ftr XXX TR
.ftr XXX YYY
.ft XXX
    ⇒ warning: can't find font 'XXX'
```

`.fzoom` *f* [*zoom*] [Request]
`\n[.zoom]` [Register]

Set magnification of font *f* to factor *zoom*, which must be a non-negative integer multiple of 1/1000th. This request is useful to adjust the optical size of a font in relation to the others. In the example below, font `CR` is magnified by 10% (the zoom factor is thus 1.1).

```
.fam P
.fzoom CR 1100
.ps 12
Palatino and \f[CR]Courier\f[]
```

A missing or zero value of *zoom* is the same as a value of 1000, which means no magnification. *f* must be a real font name, not a style.

The magnification of a font is completely transparent to GNU `troff`; a change of the zoom factor doesn't cause any effect except that the dimensions of glyphs, (word) spaces, kerns, etc., of the affected font are adjusted accordingly.

The zoom factor of the current font is available in the read-only number register '`.zoom`', in multiples of 1/1000th. It returns zero if there is no magnification.

5.17.2 Font Families

Due to the variety of fonts available, `gtroff` has added the concept of *font families* and *font styles*. The fonts are specified as the concatenation of the font family and style. Specifying a font without the family part causes `gtroff` to use that style of the current family.

Currently, fonts for the devices `-Tps`, `-Tpdf`, `-Tdvi`, `-Tlj4`, `-Tlbp`, and the X11 fonts are set up to this mechanism. By default, `gtroff` uses the Times family with the four styles '`R`', '`I`', '`B`', and '`BI`'.

This way, it is possible to use the basic four fonts and to select a different font family on the command line (see Section 2.1 [Groff Options], page 7).

`.fam` [*family*] [Request]
`\n[.fam]` [Register]
`\F`*f* [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; `\FP` 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.
```

`\F` doesn't produce an input token in GNU `troff`. 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 whose name is the concatenation of the name of the current family and the name of the current style. For example, if the current font is 1 and font position 1 is associated with style '`R`' and the current font family is '`T`', then font '`TR`' is used. If the current font is not a style, then the current family is ignored. If the requests `cs`, `bd`, `tkf`, `uf`, or `fspecial` are applied to a style, they are instead applied to the member of the current family corresponding to that style.

n must be a non-negative integer.

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 118.

5.17.3 Font Positions

For compatibility with AT&T `troff`, GNU `troff` has the concept of font *positions* at 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 GNU `troff` starts, it uses font position 1 (which must exist; position 0 is unused at start-up[27]).

The current font in use, as a font position, is available in the read-only number register '`.f`'. This can be useful to save the current font for later recall. It is associated with the current environment (see Section 5.26 [Environments], page 174).

```
.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 GNU `troff` does not enforce this strictly.

[27] Usually.

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 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 114, for the standard syntax form.

5.17.4 Using Symbols

A *glyph* is a graphical representation of a *character*. While a character is an abstract entity containing semantic information, a glyph is something that can be actually seen on screen or paper. It is possible that a character has multiple glyph representation forms (for example, the character 'A' can be either written in a roman or an italic font, yielding two different glyphs); sometimes more than one character maps to a single glyph (this is a *ligature*—the most common is 'fi').

A *symbol* is simply a named glyph. Within `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 are 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. 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` that is assumed to be missing. There are neither font-specific special fonts for `XXX` nor any other fonts made special with the `special` request, so `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 [Device and Font Files], page 222, and Section 5.17.6 [Special Fonts], page 127, for more details.

The list of available symbols is device dependent; see the *groff_char(7)* man page for a complete list of all glyphs. For example, say

```
man -Tdvi groff_char > groff_char.dvi
```

for a list using the default DVI fonts (not all versions of the man program support the -T option). If you want to use an additional macro package to change the used fonts, groff must be called directly:

```
groff -Tdvi -mec -man groff_char.7 > groff_char.dvi
```

Glyph names not listed in groff_char(7) are derived algorithmically, using a simplified version of the Adobe Glyph List (AGL) algorithm, which is described in https://github.com/adobe-type-tools/agl-aglfn. The (frozen) set of glyph names that can't be derived algorithmically is called *groff glyph list (GGL)*.

- A glyph for Unicode character U+*XXXX*[*X*[*X*]], which is not a composite character is named 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 either.
- A glyph representing more than a single input character is named

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

  Example: u0045_0302_0301.

  For simplicity, all Unicode characters that are composites must be maximally decomposed to NFD[28]; for example, u00CA_0301 is not a valid glyph name since U+00CA (LATIN CAPITAL LETTER E WITH CIRCUMFLEX) can be further decomposed into U+0045 (LATIN CAPITAL LETTER E) and U+0302 (COMBINING CIRCUMFLEX ACCENT). u0045_0302_0301 is thus the glyph name for U+1EBE, LATIN CAPITAL LETTER E WITH CIRCUMFLEX AND ACUTE.
- groff maintains a table to decompose all algorithmically derived glyph names that are composites itself. For example, u0100 (LATIN LETTER A WITH MACRON) is automatically decomposed into u0041_0304. Additionally, a glyph name of the GGL is preferred to an algorithmically derived glyph name; groff also automatically does the mapping. Example: The glyph u0045_0302 is mapped to ^E.
- glyph names of the GGL can't be used in composite glyph names; for example, ^E_u0301 is invalid.

\(*nm* [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

[28] This is "Normalization Form D" as documented in Unicode Standard Annex #15 (https://unicode.org/reports/tr15/).

syntax for one-character names—the natural form '`\n`' would collide with escapes.[29]

If *name* is undefined, a warning of type '`char`' is generated, and the escape is ignored. See Section 5.33 [Debugging], page 188, for information about warnings.

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

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

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

Examples:

`\[A ho]` '`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`.

```
\[^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*.[30] 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.

---

[29] 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, GNU `troff` 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.

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

`.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.

`` \` `` [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.

`\_` [Escape]
This is the same as `\[ul]`, the underline character.

`.cflags` *n c1 c2 ...* [Request]
Assign properties encoded by the number *n* to characters *c1*, *c2*, and so on.

Input characters, including special characters introduced by an escape, have certain properties associated with them.[31] These properties can be

[31] Output glyphs don't have such properties. For GNU `troff`, a glyph is a box numbered with an index into a font, a given height above and depth below the baseline, and a width—nothing more.

modified with this request. The first argument is the sum of the desired flags and the remaining arguments are the characters to be assigned those properties. Spaces between the *cn* arguments are optional. Any argument *cn* can be a character class defined with the `class` request rather than an individual character. See Section 5.17.5 [Character Classes], page 126.

The non-negative integer *n* is the sum of any of the following. Some combinations are nonsensical, such as '`33`' (1 + 32).

`1` Recognize the character as ending a sentence if followed by a newline or two spaces. Initially, characters '`.?!`' have this property.

`2` Enable breaks before the character. A line is not broken at a character with this property unless the characters on each side both have non-zero hyphenation codes. This exception can be overridden by adding 64. Initially, no characters have this property.

`4` Enable breaks after the character. A line is not broken at a character with this property unless the characters on each side both have non-zero hyphenation codes. This exception can be overridden by adding 64. Initially, characters '`\-\[hy]\[em]`' have this property.

`8` Mark the glyph associated with this character as overlapping other instances of itself horizontally. Initially, characters '`\[ul]\[rn]\[ru]\[radicalex]\[sqrtex]`' have this property.

`16` Mark the glyph associated with this character as overlapping other instances of itself vertically. Initially, the character '`\[br]`' has this property.

`32` Mark the character as transparent for the purpose of end-of-sentence recognition. In other words, an end-of-sentence character followed by any number of characters with this property is treated as the end of a sentence if followed by a newline or two spaces. This is the same as having a zero space factor in TEX. Initially, characters '`"')]*\[dg]\[dd]\[rq]\[cq]`' have this property.

`64` Ignore hyphenation codes of the surrounding characters. Use this in combination with values 2 and 4 (initially, no characters have this property).

For example, if you need an automatic break point after the en-dash in numerical ranges like "3000–5000", insert

```
.cflags 68 \[en]
```

into your document. Note, however, that this can lead to bad layout if done without thinking; in most situations, a better

solution instead of changing the `cflags` value is to insert `\:` right after the hyphen at the places that really need a break point.

The remaining values were implemented for East Asian language support; those who use alphabetic scripts exclusively can disregard them.

`128` Prohibit a line break before the character, but allow a line break after the character. This works only in combination with flags 256 and 512 and has no effect otherwise. Initially, no characters have this property.

`256` Prohibit a line break after the character, but allow a line break before the character. This works only in combination with flags 128 and 512 and has no effect otherwise. Initially, no characters have this property.

`512` Allow line break before or after the character. This works only in combination with flags 128 and 256 and has no effect otherwise. Initially, no characters have this property.

In contrast to values 2 and 4, the values 128, 256, and 512 work pairwise. If, for example, the left character has value 512, and the right character 128, no break will be automatically inserted between them. If we use value 6 instead for the left character, a break after the character can't be suppressed since the neighboring character on the right doesn't get examined.

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

Define a new character or glyph *g* to be *string*, which can be empty. More precisely, `char` defines a `groff` object (or redefines an existing one) that is accessed with the name *g* on input, and produces *string* on output. Every time glyph *g* needs to be printed, *string* is processed in a temporary environment and the result is wrapped up into a single object. Compatibility mode is turned off and the escape character is set to `\` while *string* is processed. Any emboldening, constant spacing, or track kerning is applied to this object rather than to individual glyphs in *string*.

An object 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 with the `lc` request; repeated patterns can be drawn with it using the `\l` and `\L` escape sequences; and words containing *g* can be hyphenated correctly if the `hcode` request is used to give the object a hyphenation code.

There is a special anti-recursion feature: use of the object within its own definition is handled like a normal character (not defined with `char`).

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 5.17.5 [Character Classes], page 126.

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

Remove the definitions of glyphs *c1*, *c2*, . . ., undoing the effect of a `char`, `fchar`, or `schar` request.

Spaces and tabs are optional between *cn* arguments.

The request `rfschar` removes glyph definitions defined with `fschar` for font *f*.

See Section 7.1 [Special Characters], page 201.

5.17.5 Character Classes

Classes are particularly useful for East Asian languages such as Chinese, Japanese, and Korean, where the number of needed characters is much larger than in European languages, and where large sets of characters share the same properties.

`.class` *name c1 c2 . . .* [Request]

Define a character class (or simply "class") *name* comprising the characters *c1*, *c2*, and so on.

A class thus defined can then be referred to in lieu of listing all the characters within it. Currently, only the `cflags` request can handle references to character classes.

In the request's simplest form, each *cn* is a character (or special character).

```
.class [quotes] ' \[aq] \[dq] \[oq] \[cq] \[lq] \[rq]
```

Since class and glyph names share the same name space, it is recommended to start and end the class name with `[` and `]`, respectively, to avoid collisions with existing character names defined by GNU `troff` or the user (with `char` and related requests). This practice applies the presence of `]` in the class name to prevent the use of the special character escape form `\[...]`, thus you must use the `\C` escape to access a class with such a name.

You can also use a character range notation consisting of a start character followed by '`-`' and then an end character. Internally, GNU `troff` converts these two symbol names to Unicode code points (according to the groff glyph list [GGL]), which then give the start and end value of the range. If that fails, the class definition is skipped.

Furthermore, classes can be nested.

```
.class [prepunct] , : ; > }
.class [prepunctx] \C'[prepunct]' \[u2013]-\[u2016]
```

The class '`[prepunctx]`' thus contains the contents of the class `[prepunct]` as defined above (the set '`, : ; > }`'), and characters in the range between `U+2013` and `U+2016`.

If you want to include '`-`' in a class, it must be the first character value in the argument list, otherwise it gets misinterpreted as part of the range syntax.

It is not possible to use class names as end points of range definitions.

A typical use of the `class` request is to control line-breaking and hyphenation rules as defined by the `cflags` request. For example, to inhibit line breaks before the characters belonging to the `prepunctx` class defined in the previous example, you can write the following.

```
.cflags 2 \C'[prepunctx]'
```

See the `cflags` request in Section 5.17.4 [Using Symbols], page 119, for more details.

5.17.6 Special Fonts

Special fonts are those that `gtroff` searches when it cannot find the requested glyph in the current font. The Symbol font is usually a special font.

`gtroff` provides the following two requests to add more special fonts. See Section 5.17.4 [Using Symbols], page 119, 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 that appear in a call to `special` or `fspecial` are loaded.

See Section 5.17.4 [Using Symbols], page 119, for the exact search order of glyphs.

5.17.7 Artificial Fonts

There are a number of requests and escapes for artificially creating fonts. These are largely vestiges of the days when output devices did not have a wide variety of fonts, and when `nroff` and `troff` were separate programs. Most of them are no longer necessary in GNU `troff`. Nevertheless, they are supported.

`\H'`*height*`'` [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` and `-Tpdf` devices support this feature.

`\H` doesn't produce an input token in GNU `troff`. 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` and `-Tpdf` devices support this feature.

`\S` doesn't produce an input token in GNU `troff`. 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 174). 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 request 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.

5.17.8 Ligatures and Kerning

Ligatures are groups of characters that are run together, i.e, producing a single glyph. For example, the letters 'f' and 'i' can form a ligature 'fi' as in the word 'file'. This produces a cleaner look (albeit subtle) to the printed output. Usually, ligatures are not available in fonts for TTY output devices.

Most PostScript fonts support the fi and fl ligatures. The C/A/T typesetter that was the target of AT&T `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 225.

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*.

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.

Sometimes, when typesetting letters of different fonts, more or less space at such boundaries is 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 non-printing input break, which is invisible. Its intended use is to stop interaction of a character with its surroundings.

- 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
    error  warning: macro '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 101).

`\)` [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

GNU `troff` uses two dimensions with each line of text, type size and vertical spacing. The *type size* is approximately the height of the tallest glyph.[32] *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.

Typesetters call the difference between type size and vertical spacing *leading*.[33]

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. `ps` with no argument restores the previous size.

---

[32] This is usually the parenthesis. 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', '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.

[33] This is pronounced to rhyme with "sledding", and refers to the use of lead metal (Latin: *plumbum*) in traditional typesetting.

The default scaling indicator of `size` is 'z'. If the resulting size is non-positive, 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 (see Section 5.18.2 [Fractional Type Sizes], page 136).

`.s` is associated with the current environment (see Section 5.26 [Environments], page 174).

```
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 is valid:

`\sn` Set the point size to *n* points. *n* must be a single digit. If *n* is 0, restore the previous size.

`\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.

See Section 5.18.2 [Fractional Type Sizes], page 136, for additional syntactical forms of the `\s` escape (which accept integers as well as fractions).

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]
```

`.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).

'.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 174).

The effective vertical line spacing consists of four components. Breaking a line causes the following actions (in the given order).

- Move the current point vertically by the *extra pre-vertical line space*. This is the minimum value of all \x escapes with a negative argument in the current output line.
- Move the current point vertically by the vertical line spacing as set with the vs request.
- Output the current line.
- Move the current point vertically by the *extra post-vertical line space*. This is the maximum value of all \x escapes with a positive argument in the line that has just been output.
- Move the current point vertically by the *post-vertical line spacing* as set with the pvs request.

It is usually better to use vs or pvs instead of ls to produce double-spaced documents: vs and pvs have a finer granularity for the inserted vertical space than ls; furthermore, certain preprocessors assume single spacing.

See Section 5.9 [Manipulating Spacing], page 95, 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 174).

5.18.2 Fractional Type Sizes

A *scaled point* is equal to 1/*sizescale* points, where *sizescale* is specified in the device description file `DESC`, and defaults to 1 . A new scale indicator '`z`' has has the effect of multiplying by *sizescale*. Requests and escape sequences in GNU `troff` interpret arguments that represent a point size as being in units of scaled points; that is, they evaluate each such argument using a default scale indicator of '`z`'. Arguments treated in this way comprise those to the escapes `\H` and `\s`, to the request `ps`, the third argument to the `cs` request, and the second and fourth arguments to the `tkf` request.

For example, if *sizescale* is 1000, then a scaled point is one one-thousandth of a point. The request '`.ps 10.25`' is synonymous with '`.ps 10.25z`' and sets the point size to 10250 scaled points, or 10.25 points.

Consequently, in GNU `troff`, the number register `.s` can contain a non-integral point size.

It makes no sense to use the '`z`' scale indicator in a numeric expression whose default scale indicator is neither '`u`' nor '`z`', so GNU `troff` disallows this. Similarly, it is nonsensical to use a scaling indicator other than '`z`' or '`u`' in a numeric expression whose default scale indicator is '`z`', and so GNU `troff` disallows this as well.

Another new scale indicator '`s`' multiplies by the number of basic units in a scaled point. For instance, '`\n[.ps]s`' is equal to '`1m`' by definition. Do not 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 174).

`\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.

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 174).

The `\s` escape has the following syntax for working with fractional type sizes:

`\s[`*n*`]`
`\s'`*n*`'` Set the point size to *n* scaled points; *n* is a numeric expression with a default scale indicator of '`z`'.

`\s[+`*n*`]`
`\s[-`*n*`]`
`\s+[`*n*`]`
`\s-[`*n*`]`
`\s'+`*n*`'`
`\s'-`*n*`'`
`\s+'`*n*`'`
`\s-'`*n*`'` Increase or decrease the point size by *n* scaled points; *n* is a numeric expression (which may start with a minus sign) with a default scale indicator of '`z`'.

See Section 8.2 [Device and Font Files], page 222.

5.19 Strings

GNU `troff` has string variables primarily for user convenience. Only one string is predefined by the language.

`\*[.T]` [String]
Contains the name of the output driver (for example, '`utf8`' or '`pdf`').

The `ds` (or `ds1`) request creates a string with a specified name and contents and the `\*` escape dereferences its name, retrieving the contents. Dereferencing an undefined string name defines it as empty.

`.ds` *name* [*string*] [Request]
`.ds1` *name* [*string*] [Request]
`\*`*n* [Escape]
`\*(`*nm* [Escape]
`\*[`*name* [*arg1 arg2 . . .*]`]` [Escape]
Define a string variable *name* with contents *string*. If *name* already exists, it is removed first (see `rm` below). The syntax form using brackets accepts arguments that are handled as macro arguments are; recall Section 5.5.1.1 [Request and Macro Arguments], page 72. In contrast to macro invocations, however, a closing bracket as a string argument must be enclosed in double quotes.

The `\*` escape *interpolates* (expands in place) a previously defined string variable *name* (one-character name *n*, two-character name *nm*). More precisely, the stored string is pushed onto the input stack, which is then parsed normally. Similarly to number registers, it is possible to nest strings; i.e., string variables can be called within string variables. An argument in a string definition must be escaped for correct behavior; See Section 5.21.2 [Parameters], page 152.

```
.ds a \\$1 wildebeest
.ds b big, \*[a hairy]
I see a \*[b].
    ⇒ I see a big, hairy wildebeest.
```

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 188).

If `ds` is called with only one argument, *name* is defined as an empty string.

Caution: Unlike other requests, the second argument to the `ds` request consumes the remainder of the input line, including trailing spaces. This means that comments on a line with such a request can introduce unwanted space into a string when they are set off from the material they annotate, as is conventional.

```
.ds TeX T\h'-.2m'\v'.2m'E\v'-.2m'\h'-.1m'X \" Knuth's TeX
```

Instead, place the comment on another line or put the comment escape immediately adjacent to the last character of the string.

```
.ds TeX T\h'-.2m'\v'.2m'E\v'-.2m'\h'-.1m'X\" Knuth's TeX
```

It is good style to end string definitions (and appendments; see below) with a comment, even an empty one, to prevent unwanted space from creeping into them during source document maintenance.

```
.ds author Alice Pleasance Liddell\"
.ds empty \" might be appended to later with .as
```

To store leading space in a string, start it with a double quote. A double quote is special only in that position; double quotes in any other location are included in the string (the effects of escape sequences notwithstanding).

```
.ds salutation "         Yours in a white wine sauce,\"
.ds c-var-defn "    char build_date[]="2020-07-29";\"
.ds sucmd sudo sh -c "fdisk -l /dev/sda > partitions"\"
```

Strings are not limited to a single line of input text. A string can span several lines by escaping the newlines with a backslash. The resulting string is stored *without* the newlines.

```
.ds foo This string contains \
text on multiple lines \
of input.
```

It is not possible to embed a newline in a string that will be interpreted as such when the string is interpolated. To achieve that effect, use the `\*` escape to interpolate a macro instead.

Strings, macros, and diversions (and boxes) share the same name space. Internally, the same mechanism is used to store them. Thus it is possible to invoke a macro with string interpolation syntax and vice versa.

```
.de subject
Typesetting
..
.de predicate
rewards attention to detail
..
\*[subject] \*[predicate].
Truly.
    ⇒ Typesetting
    ⇒  rewards attention to detail Truly.
```

What went wrong? Strings don't contain newlines, but macros do. String interpolation placed a newline at the end of '`\*[subject]`', and the next thing on the input was a space. Similarly, when '`\*[predicate]`' was interpolated, it was followed by the empty request '`.`' on a line by itself. If we want to use macros as strings, we must take interpolation behavior into account.

```
.de subject
Typesetting\\
..
.de predicate
rewards attention to detail\\
..
\*[subject] \*[predicate].
Truly.
    ⇒ Typesetting rewards attention to detail.  Truly.
```

By ending each text line of the macros with an escaped '`\RET`', we get the desired effect (see Section 5.14 [Line Control], page 109). What would have happened if we had used only one backslash at a time instead?

Interpolating a string does not hide existing macro arguments. Thus in a macro, a more efficient way of doing

```
.xx \\$@
```

is

```
\\*[xx]\\
```

The latter calling syntax doesn't change the value of `\$0`, which is then inherited from the calling macro (see Section 5.21.2 [Parameters], page 152).

Diversions and boxes can be also called with string syntax. It is sometimes convenient to 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 subsequent newline from being interpreted as a break (again, see Section 5.14 [Line Control], page 109).

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 AT&T `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 186.

The `ds1` request defines a string such that compatibility mode is off when the string is later interpolated. To be more precise, a *compatibility save* input token is inserted at the beginning of the string, and a *compatibility restore* input token at the end.

```
.nr xxx 12345
.ds aa The value of xxx is \\n[xxx].
.ds1 bb The value of xxx is \\n[xxx].
.
.cp 1
.
\*(aa
     error  warning: number register '[' not defined
    ⇒ The value of xxx is 0xxx].
\*(bb
    ⇒ The value of xxx is 12345.
```

`.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. If `as` is called with only one argument, no operation is performed (beyond dereferencing it).

```
.as salutation " with shallots, onions and garlic,\"
```

The `as1` request is similar to `as`, but compatibility mode is switched off when the appended portion of the string is later interpolated. 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.

Several requests exist to perform rudimentary string operations. Strings can be queried (`length`) and modified (`chop`, `substring`, `stringup`, `stringdown`), and their names can be manipulated through renaming, removal, and aliasing (`rn`, `rm`, `als`).

`.length` *reg anything* [Request]

Compute the number of characters of *anything* and store the count in the number register *reg*. If *reg* doesn't exist, it is created. *anything* is read in copy mode.

```
.ds xxx abcd\h'3i'efgh
.length yyy \*[xxx]
\n[yyy]
    ⇒ 14
```

`.chop` *object* [Request]

Remove the last character from the macro, string, or diversion named *object*. This is useful for removing the newline from the end of a diversion that is to be interpolated as a string. This request can be used repeatedly on the same *object*; see Section 5.32 [Gtroff Internals], page 186, for details on nodes inserted additionally by GNU `troff`.

`.substring` *str start* [*end*] [Request]

Replace the string named *str* with its substring bounded by the indices *start* and *end*, inclusive. The first character in the string has index 0. If

end is omitted, it is implicitly set to the largest valid value (the string length minus one). Negative indices count backwards from the end of the string: the last character has index −1, the character before the last has index −2, and so on.

```
.ds xxx abcdefgh
.substring xxx 1 -4
\*[xxx]
    ⇒ bcde
.substring xxx 2
\*[xxx]
    ⇒ de
```

`.stringdown` *str* [Request]
`.stringup` *str* [Request]

Alter the string named *str* by replacing each of its bytes with its lowercase (`stringdown`) or uppercase (`stringup`) version (if one exists). GNU `troff` special characters (see the *groff_char(7)* man page) can be used and the output will usually transform in the expected way due to the regular naming convention of the special character escapes.

```
.ds resume R\['e]sum\['e]
\*[resume]
.stringdown resume
\*[resume]
.stringup resume
\*[resume]
    ⇒ Résumé résumé RÉSUMÉ
```

(In pratice, we would end the `ds` request with a comment escape `\"` to prevent whitespace from creeping into the definition during source document maintenance.)

`.rn` *old new* [Request]

Rename the request, macro, diversion, or string *old* to *new*.

`.rm` *name* [Request]

Remove the request, macro, diversion, or string *name*. GNU `troff` treats subsequent invocations as if the name had never been defined.

`.als` *new old* [Request]

Create an alias *new* for the existing request, string, macro, or diversion object named *old*, causing the names to refer to the same stored object. If *old* is undefined, a warning of type '`mac`' is generated and the request is ignored.

To understand how the `als` request works, consider two different storage pools: one for objects (macros, strings, etc.), and another for names. As soon as an object is defined, GNU `troff` adds it to the object pool, adds its name to the name pool, and creates a link between them. When `als`

creates an alias, it adds a new name to the name pool that gets linked to the same object as the old name.

Now consider this example.

```
.de foo
..
.
.als bar foo
.
.de bar
.  foo
..
.
.bar
error  input stack limit exceeded
error  (probable infinite loop)
```

In the above, `bar` remains an *alias*—another name for—the object referred to by `foo`, which the second `de` request replaces. Alternatively, imagine that the `de` request *dereferences* its argument before replacing it. Either way, the result of calling `bar` is a recursive loop that finally leads to an error. See Section 5.21 [Writing Macros], page 148.

To remove an alias, simply call `rm` on its name. The object itself is not destroyed until it has no more names.

5.20 Conditionals and Loops

GNU `troff` has `if` and `while` control structures like other languages. However, the syntax for grouping multiple input lines in the branches or bodies of these structures is unusual.

5.20.1 Operators in Conditionals

In `if`, `ie`, and `while` requests, in addition to ordinary numeric expressions (see Section 5.3 [Expressions], page 67), several boolean operators are available.

`c` *glyph*
: True if a *glyph* is available, where *glyph* is a Unicode basic Latin character, a GNU `troff` special character '`\(`*xx*' or '`\[`*xxx*`]`', '`\N'`*xxx*`'`', or has been defined by the `char` request.

`d` *name*
: True if there is a string, macro, diversion, or request called *name*.

`e`
: True if the current page is even-numbered.

`F` *font*
: True if a font called *font* exists. *font* is handled as if it were opened with the `ft` request (that is, font translation and styles are applied), without actually mounting it.

  This test doesn't load the complete font, but only its header to verify its validity.

`m` *color* — True if there is a color called *color*.

`n` — True if the document is being processed in `nroff` mode (i.e., the `nroff` request has been issued). See Section 5.12 [Troff and Nroff Mode], page 106.

`o` — True if the current page is odd-numbered.

`r` *reg* — True if there is a number register called *reg*.

`S` *style* — True if a style called *style* has been registered. Font translation is applied.

`t` — True if the document is being processed in `troff` mode (i.e., the `troff` request has been issued). See Section 5.12 [Troff and Nroff Mode], page 106.

`v` — Always false. This condition is recognized only for compatibility with certain other `troff` implementations.[34]

'*xxx*'*yyy*'

True if the output produced by *xxx* is equal to the output produced by *yyy*. Other characters can be used in place of the single quotes; the same set of delimiters as for the `\D` escape is used (see Section 5.5.3 [Escapes], page 73). `gtroff` formats *xxx* and *yyy* in separate environments; after the comparison the resulting data is discarded.

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

The resulting motions, glyph sizes, and fonts have to match,[35] 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.

To compare strings without processing, surround the data with `\?`.

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

[34] This refers to `vtroff`, a translator that would convert the C/A/T output from early-vintage AT&T `troff` to a form suitable for Versatec and Benson-Varian plotters.

[35] The created output nodes must be identical. See Section 5.32 [Gtroff Internals], page 186.

Since data protected with \? is read in copy mode it is even possible to use incomplete input without causing an error.

```
.ds a \[
.ds b \[
.ie '\?\*a\?'\?\*b\?' \
true
.el \
false
    ⇒ true
```

These operators can't be combined with other operators like '`:`' or '`&`'; only a leading '`!`' (without spaces or tabs between the exclamation mark and the operator) can be used to negate the result.

```
.nr x 1
.ie !r x register x is not defined
.el      register x is defined
    ⇒ register x is defined
```

Spaces and tabs immediately after '`!`' cause the condition to evaluate as zero (this bizarre behavior maintains compatibility with AT&T `troff`).

```
.nr x 1
.ie ! r x register x is not defined
.el       register x is defined
    ⇒ r x register x is not defined
```

The unexpected appearance of '`r x`' in the output is a clue that our conditional was not interpreted the way we planned, but matters may not always be so obvious.

Spaces and tabs are optional before the arguments to the '`r`', '`d`', and '`c`' operators.

5.20.2 if-then

`.if` *expr anything* [Request]

Evaluate the expression *expr*, and execute *anything* (the remainder of the line) if *expr* evaluates true (that is, to a value greater than zero). *anything* is interpreted as though it were on a line by itself (except that leading spaces are ignored). See Section 5.20.1 [Operators in Conditionals], page 143.

```
.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`'.

5.20.3 if-else

`.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
```

See Section 5.3 [Expressions], page 67.

5.20.4 Conditional Blocks

`\{` [Escape]
`\}` [Escape]

It is frequently desirable for a control structure to govern more than one request, call more than one macro, span more than one input line of text, or mix the foregoing. The opening and closing brace escapes `\{` and `\}` perform such grouping. Brace escapes can be used outside of control structures, but when they are they have no meaning and produce no output.

`\{` should appear (after optional spaces and tabs) immediately subsequent to the request's conditional expression. `\}` should appear on a line with other occurrences of itself as necessary to match `\{` escapes. It can be preceded by a control character, spaces, and tabs. Input after an `\}` escape on the same line is only processed if all the preceding conditions to which the escapes correspond are true. Furthermore, a `\}` closing the body of a `while` request must be the last such escape on an input line.

```
A
.if 0 \{ B
C
D
\}E
F
    ⇒ A F
N
.if 1 \{ O
.  if 0 \{ P
Q
R\} S\} T
U
    ⇒ N O U
```

If the above behavior challenges the intuition, keep in mind that it was implemented to retain compatibility with AT&T `troff`. For clarity, it is common practice to end input lines with `\{`, optionally followed by `\RET`

to suppress a break before subsequent text lines, and to have nothing more than a control character, spaces, and tabs before any lines containing \}.

```
.de DEBUG
debug =
.ie \\$1 \{\
ON,
development
\}
.el \{\
OFF,
production
\}
version
..
.DEBUG 0
.br
.DEBUG 1
```

Try omitting the \RETs from the foregoing example and see how the output changes. Remember that, as noted above, after a true conditional (or after the el request if its counterpart ie condition was false) any spaces or tabs on the same input line are interpreted *as if they were on an input line by themselves.*

5.20.5 while

GNU troff provides a looping construct using the while request, which is used much like the if request.

.while *expr anything* [Request]

Evaluate the expression *expr*, and repeatedly execute *anything* (the remainder of the line) until *expr* evaluates false.

```
.nr a 0 1
.while (\na < 9) \{\
\n+a,
.\}
\n+a
    ⇒ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
```

Some remarks.

- The body of a while request is treated like the body of a de request: gtroff temporarily stores it in a macro that is deleted after the loop has been exited. It can considerably slow down a macro if the body of the while request (within the macro) is large. Each time the macro is executed, the while body is parsed and stored again as a temporary macro.

```
.de xxx
.  nr num 10
.  while (\\n[num] > 0) \{\
.    \" many lines of code
.    nr num -1
.  \}
..
```

The traditional and often better solution (AT&T `troff` lacked the `while` request) is to use a recursive macro instead that is parsed only once during its definition.

```
.de yyy
.  if (\\n[num] > 0) \{\
.    \" many lines of code
.    nr num -1
.    yyy
.  \}
..
.
.de xxx
.  nr num 10
.  yyy
..
```

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.

5.21 Writing Macros

A *macro* is a collection of text and embedded commands that can be invoked multiple times. Use macros to define common operations. See Section 5.19 [Strings], page 137, for a (limited) alternative syntax to call macros.

Although the following requests can be used to create macros, simply using an undefined macro will cause it to be defined as empty. See Section 5.4 [Identifiers], page 69.

`.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). If the optional second argument to `de` is present it is used as the macro closure request instead of '`..`'.

There can be spaces or tabs after the first dot in the line containing the ending token (either '`.`' or macro '*end*'). Don't insert a tab character immediately after the '`..`', otherwise it isn't recognized as the end-of-macro symbol.[36]

Here is a small example macro called '`P`' that causes a break and inserts some vertical space. It could be used to separate paragraphs.

```
.de P
.  br
.  sp .8v
..
```

The following example defines a macro within another. Remember that expansion must be protected twice; once for reading the macro and once for executing.

```
\# a dummy macro to avoid a warning
.de end
..
.
.de foo
.  de bar end
.    nop \f[B]Hello \\\\$1!\f[]
.  end
..
.
.foo
.bar Joe
```

⇒ **Hello Joe!**

[36] While it is possible to define and call a macro '`.`' with

```
.de .
.  tm foo
..
.
..    \" This calls macro '.'!
```

you can't use this as the end-of-macro macro: during a macro definition, '`..`' is never handled as a call to '`.`', even if you say '`.de foo .`' explicitly.

Since `\f` has no expansion, it isn't necessary to protect its backslash. Had we defined another macro within `bar` that takes a parameter, eight backslashes would be necessary before '`$1`'.

The `de1` request turns off compatibility mode while executing the macro. On entry, the current compatibility mode is saved and restored at exit.

```
.nr xxx 12345
.
.de aa
The value of xxx is \\n[xxx].
..
.de1 bb
The value of xxx is \\n[xxx].
..
.
.cp 1
.
.aa
    ⇒ warning: number register ‘[’ not defined
    ⇒ The value of xxx is 0xxx].
.bb
    ⇒ The value of xxx is 12345.
```

The `dei` request defines a macro indirectly. That is, it expands strings whose names are *name* or *end* before performing the append.

This:

```
.ds xx aa
.ds yy bb
.dei xx yy
```

is equivalent to:

```
.de aa bb
```

The `dei1` request is similar to `dei` but with compatibility mode switched off during execution of the defined macro.

If compatibility mode is on, `de` (and `dei`) behave similar to `de1` (and `dei1`): A 'compatibility save' token is inserted at the beginning, and a 'compatibility restore' token at the end, with compatibility mode switched on during execution. See Section 5.32 [Gtroff Internals], page 186, 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`.

Macro identifiers share their name space with identifiers for strings and diversions (and boxes).

See [the description of the `als` request], page 142, for possible pitfalls if redefining a macro that has been aliased.

`.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 set indented instead of block paragraphs, add the necessary code to the existing macro.

```
.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 137, for the `als` and `rn` request to create an alias and rename a macro, respectively.

The `am`, `as`, `da`, `de`, `di`, and `ds` requests (together with their variants) only create a new object if the name of the macro, diversion, or string is currently undefined or if it is defined as 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 Mode

When GNU `troff` processes certain requests, most importantly those which define a macro, string, or diversion, it does so in *copy mode*: it copies the characters of the definition into a dedicated storage region, interpolating the escape sequences `\n`, `\$`, and `\*`, intepreting `\\` and `\RET` immediately and storing all other escape sequences in an encoded form.

Since the escape character escapes itself, you can control whether any escape sequence is interpreted at definition time or when it is later invoked or interpolated by selectively insulating the escapes with an extra backslash.[37]

[37] Compare this to the `\def` and `\edef` commands in TEX.

```
.nr x 20
.de y
.nr x 10
\&\nx
\&\\nx
..
.y
    ⇒ 20 10
```

The counterpart to copy mode—a `roff` program's behavior when not defining a macro, string or diversion—where escapes are interpolated, requests invoked, and macros called immediately upon recognition, can be termed *interpretation mode*.

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.

  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 than or equal to 10), and the third any positive integer value (larger than zero). Macros and strings can have an unlimited number of arguments. Because string and macro definitions are read in copy mode, use two backslashes on these in practice to prevent their 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 72).

`\$^` [Escape]

Handle the parameters of a macro as if they were an argument to the `ds` or similar requests.

```
.de foo
.  tm $1=‘\\$1’
.  tm $2=‘\\$2’
.  tm $*=‘\\$*’
.  tm $@=‘\\$@’
.  tm $^=‘\\$^’
..
.foo " This is a "test"
    ⇒ $1=‘ This is a ’
    ⇒ $2=‘test"’
    ⇒ $*=‘ This is a  test"’
    ⇒ $@=‘" This is a " "test""’
    ⇒ $^=‘" This is a "test"’
```

This escape is useful mainly for macro packages like `trace.tmac`, which redefines some requests and macros for debugging purposes.

`\$0` [Escape]

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

If a macro is called as a string (within another macro), the value of `\$0` isn't changed.

```
.de foo
.  tm \\$0
..
.als foo bar
.
```

```
.de aaa
.  foo
..
.de bbb
.  bar
..
.de ccc
\\*[foo]\\
..
.de ddd
\\*[bar]\\
..
.
.aaa
    ⇒ foo
.bbb
    ⇒ bar
.ccc
    ⇒ ccc
.ddd
    ⇒ ddd
```

See Section 5.5.1.1 [Request and Macro Arguments], page 72.

5.22 Page Motions

See Section 5.9 [Manipulating Spacing], page 95, 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`'.

If a page break occurs between a `mk` request and its matching `rt` request, the `rt` is silently ignored.

Here a primitive solution for a two-column macro.

```
.nr column-length 1.5i
.nr column-gap 4m
.nr bottom-margin 1m
.


.de 2c
.  br
.  mk
.  ll \\n[column-length]u
.  wh -\\n[bottom-margin]u 2c-trap
.  nr right-side 0
..
.


.de 2c-trap
.  ie \\n[right-side] \{\
.    nr right-side 0
.    po -(\\n[column-length]u + \\n[column-gap]u)
.    \" remove trap
.    wh -\\n[bottom-margin]u
.  \}
.  el \{\
.    \" switch to right side
.    nr right-side 1
.    po +(\\n[column-length]u + \\n[column-gap]u)
.    rt
.  \}
..
.


.pl 1.5i
.ll 4i
This is a small test that shows how the
rt request works in combination with mk.

.2c
Starting here, text is typeset in two columns.
Note that this implementation isn't robust
and thus not suited for a real two-column
macro.
```

Result:

```
This is a small test that shows how the
rt request works in combination with mk.

Starting here,    isn't    robust
text is typeset   and   thus  not
in two columns.   suited  for   a
Note that  this   real two-column
implementation    macro.
```

The following escapes give fine control of movements about the page.

`\v'`*e*`'` [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 that prints a marker in the margin to indicate continuation of a footnote or something similar.

There are some special-case escapes for vertical motion.

`\r` [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 unbreakable space. Ignored for TTY output devices (rounded to zero).

However, if there is a glyph defined in the current font file with name `\|` (note the leading backslash), the width of this glyph is used instead (even for TTYs).

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

However, if there is a glyph defined in the current font file with name `\^` (note the leading backslash), the width of this glyph is used instead (even for TTYs).

`\0` [Escape]
An unbreakable 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. In 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.

`\z`*g* [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 154), 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 199, and Section 6.4 [ggrn], page 199.

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 \&.

```
.de box
\[br]\\$*\[br]\l'|0\[rn]'\l'|0\[ul]'
..
```

This above 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 is the result, produced with grotty.

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

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

The `\D` escape provides a variety of drawing functions. 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'F`*x* `...'`', 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 right and down, respectively. The end point of the line is the new current location.

The following example is a macro for creating a box around a text string; for simplicity, the box margin is taken as a fixed value, 0.2 m.

```
.de BOX
.  nr @wd \w'\\$1'
\h'.2m'\
\h'-.2m'\v'(.2m - \\n[rsb]u)'\
\D'l 0 -(\\n[rst]u - \\n[rsb]u + .4m)'\
\D'l (\\n[@wd]u + .4m) 0'\
\D'l 0 (\\n[rst]u - \\n[rsb]u + .4m)'\
\D'l -(\\n[@wd]u + .4m) 0'\
\h'.2m'\v'-(.2m - \\n[rsb]u)'\
\\$1\
\h'.2m'
..
```

First, the width of the string is stored in register `@wd`. Then, four lines are drawn to form a box, properly offset by the box margin. The registers `rst` and `rsb` are set by the `\w` escape, containing the largest height and depth of the whole string.

`\D'c` *d*`'` Draw a circle with a diameter of *d* with the leftmost point at the current position. After drawing, the current location is positioned at the rightmost point of the circle.

`\D'C` *d*`'` Draw a solid circle with the same parameters and behaviour as an outlined circle. No outline is drawn.

`\D'e x y'` Draw an ellipse with a horizontal diameter of *x* and a vertical diameter of *y* with the leftmost point at the current position. After drawing, the current location is positioned at the rightmost point of the ellipse.

`\D'E x y'` Draw a solid ellipse with the same parameters and behaviour as an outlined ellipse. No outline is drawn.

`\D'a dx1 dy1 dx2 dy2'`
Draw an arc clockwise from the current location through the two specified relative locations (*dx1*,*dy1*) and (*dx2*,*dy2*). The coordinates of the first point are relative to the current position, and the coordinates of the second point are relative to the first point. After drawing, the current position is moved to the final point of the arc.

`\D'~ dx1 dy1 dx2 dy2 ...'`
Draw a spline from the current location to the relative point (*dx1*,*dy1*) and then to (*dx2*,*dy2*), and so on. The current position is moved to the terminal point of the drawn curve.

`\D'f n'` Set the shade of gray to be used for filling solid objects to *n*; *n* must be an integer between 0 and 1000, where 0 corresponds solid white and 1000 to solid black, and values in between correspond to intermediate shades of gray. This applies only to solid circles, solid ellipses, and solid polygons. By default, a level of 1000 is used.

Nonintuitively, the current point is moved horizontally to the right by *n*.

Don't use this command! It has the serious drawback that it is always rounded to the next integer multiple of the horizontal resolution (the value of the `hor` keyword in the `DESC` file). Use `\M` (see Section 5.28 [Colors], page 177) 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. The box must be drawn before the text since

colors in GNU `troff` are not transparent; the filled polygon would hide the text completely.

```
.de BOX
.  nr @wd \w'\\$1'
\h'.2m'\
\h'-.2m'\v'(.2m - \\n[rsb]u)'\
\M[lightcyan]\
\D'P 0 -(\\n[rst]u - \\n[rsb]u + .4m) \
     (\\n[@wd]u + .4m) 0 \
     0 (\\n[rst]u - \\n[rsb]u + .4m) \
     -(\\n[@wd]u + .4m) 0'\
\h'.2m'\v'-(.2m - \\n[rsb]u)'\
\M[]\
\\$1\
\h'.2m'
..
```

If you want a filled polygon that has exactly the same size as an unfilled one, you must draw both an unfilled and a filled polygon. A filled polygon is always smaller than an unfilled one because the latter uses straight lines with a given line thickness to connect the polygon's corners, while the former simply fills the area defined by the coordinates.

```
\h'1i'\v'1i'\
\# increase line thickness
\Z'\D't 5p''\
\# draw unfilled polygon
\Z'\D'p 3 3 -6 0''\
\# draw filled polygon
\Z'\D'P 3 3 -6 0''
```

`\D't` *n*`'` Set the current line thickness to *n* machine units. A value of zero selects the smallest available line thickness. A negative value makes the line thickness proportional to the current point size (this is the default behaviour of AT&T `troff`).

Nonintuitively, the current point is moved horizontally to the right by *n*.

`\D'F`*scheme 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 177); 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 214.

`\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. GNU `troff` separates the glyphs vertically by 1 m, and the whole object is centered 0.5 m above the current baseline; the largest glyph width is used as the width for the whole object. This rather inflexible positioning algorithm doesn't work with `-Tdvi` since the bracket pieces vary in height for this device. Instead, use the `eqn` preprocessor.

See Section 5.9 [Manipulating Spacing], page 95, how to adjust the vertical spacing with the `\x` escape.

5.24 Traps

Traps are locations that, when reached, call a specified macro. These traps can occur at a given location on the page, at a given location in the current diversion, at a blank line, after a certain number of input lines, or at the end of input.

Setting a trap is also called *planting*. It is also said that a trap is *sprung* if the associated macro is executed.

5.24.1 Page Location Traps

Page location traps perform an action when `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` request, or by `dt` within a diversion. 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.

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'; values of *dist* are always rounded to be multiples of the vertical resolution (as given in register `.V`).

macro is the name of the macro to execute when the trap is sprung. If *macro* is missing, remove the first trap (if any) at *dist*.

The following is a simple example of how many macro packages set headers and footers.

```
.de hd                  \" Page header
'  sp .5i
.  tl 'Title''date'
'  sp .3i
..
.
.de fo                  \" Page footer
'  sp 1v
.  tl ''%''
'  bp
..
.
.wh 0   hd              \" trap at top of the page
.wh -1i fo              \" trap one inch from bottom
```

A trap at or below the bottom of the page is ignored; it can be made active by either moving it up or increasing the page length so that the trap is on the page.

Negative trap values always use the *current* page length; they are not converted to an absolute vertical position:

```
.pl 5i
.wh -1i xx
.ptr
    ⇒ xx      -240
.pl 100i
.ptr
    ⇒ xx      -240
```

It is possible to have more than one trap at the same location; to do so, the traps must be defined at different locations, then moved together with the `ch` request; otherwise the second trap would replace the first one. Earlier defined traps hide later defined traps if moved to the same position (the many empty lines caused by the `bp` request are omitted in the following example):

```
.de a
.  nop a
..
.de b
.  nop b
..
.de c
.  nop c
..
.
.wh 1i a
.wh 2i b
.wh 3i c
.bp
    ⇒ a b c
.ch b 1i
.ch c 1i
.bp
    ⇒ a
.ch a 0.5i
.bp
    ⇒ a b
```

`\n[.t]` [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 that 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 112.

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 from an `sp` request by the most recently sprung vertical position trap, or, if the trap was sprung by an `ne` request, minus the amount of vertical motion produced by the `ne` request. In other words, at the point a trap is sprung, it represents the difference of what the vertical position would have been but for the trap, and what the vertical position actually is.

Since the `.trunc` register is only set by traps it doesn't make much sense to use it outside of trap macros.

`\n[.pe]` [Register]
A read-only register that is set to 1 while a page is ejected with the `bp` request (or by the end of input).

Outside of traps this register is always zero. In the following example, only the second call to `x` is caused by `bp`.

```
.de x
\&.pe=\\n[.pe]
.br
..
.wh 1v x
.wh 4v x
A line.
.br
Another line.
.br
    ⇒ A line.
       .pe=0
       Another line.

       .pe=1
```

An important fact to consider while designing macros is that diversions and traps do not interact normally. For example, if a trap invokes a header macro (while outputting a diversion) that tries to change the font on the current page, the effect is not visible before the diversion has completely been printed (except for input protected with `\!` or `\?`) since the data in the diversion is already formatted. In most cases, this is not the expected behaviour.

5.24.2 Diversion Traps

`.dt` [*dist macro*] [Request]
Set a trap *within* a diversion. *dist* is the location of the trap (as with the `wh` request, the default scaling indicator is '`v`') and *macro* is the name of the macro to be invoked. If called with fewer than two arguments, the diversion trap is removed.

There exists only a single diversion trap.

The number register .t still works within diversions. See Section 5.25 [Diversions], page 170.

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 that may be read before springing the trap, *macro* is the macro to be invoked. Request lines are not counted as input lines.

For example, one possible use is to have a macro that prints the next *n* lines in a bold font.

```
.de B
.  it \\$1 B-end
.  ft B
..
.
.de B-end
.  ft R
..
```

The itc request is identical except that an interrupted text line (ending with \c) is not counted as a separate line.

Both requests are associated with the current environment (see Section 5.26 [Environments], page 174); 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. If a blank line macro is thus defined, GNU troff executes *macro* when a blank line is encountered in the input file, instead of the usual behavior (see Section 5.1.4 [Breaking], page 58). If no argument is supplied, the default blank line behavior is (re-)asserted.

5.24.5 Leading Spaces Traps

.lsm *macro* [Request]
\n[lsn] [Register]
\n[lss] [Register]

Set a leading spaces trap. gtroff executes *macro* when it encounters leading spaces in an input line; the implicit line break that normally happens in this case is suppressed. A line consisting of spaces only, however, is treated as an empty line, possibly subject to an empty line macro set with the blm request.

Leading spaces are removed from the input line before calling the leading spaces macro. The number of removed spaces is stored in register `lsn`; the horizontal space that would be emitted if there was no leading space macro is stored in register `lss`. Note that `lsn` and `lss` are available even if no leading space macro has been set.

The first thing a leading space macro sees is a token. However, some escapes like `\f` or `\m` are handled on the fly (see Section 5.32 [Gtroff Internals], page 186, for a complete list) without creating a token at all. Consider that a line starts with two spaces followed by `\fIfoo`. While skipping the spaces `\fI` is handled too so that groff's current font is properly set to '`I`', but the leading space macro only sees `foo`, without the preceding `\fI`. If the macro should see the font escape you have to 'protect' it with something that creates a token, for example with `\&\fIfoo`.

5.24.6 End-of-input Traps

`.em` *macro* [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
\c
.  ne 3v
.  sp (\\n[.t]u - 3v)
.  in +4i
.  lc _
.  br
Approved:\t\a
.  sp
Date:\t\t\a
..
.
.em approval
```

The `\c` in the above example needs explanation. For historical reasons (and for compatibility with AT&T `troff`), the end macro exits as soon as it causes a page break and no remaining data is in the partially collected line.

Let us assume that there is no `\c` in the above `approval` macro, and that the page is full and has been ended with, say, a `br` request. The `ne` request now causes the start of a new page, which in turn makes `troff` exit immediately for the reasons just described. In most situations this is not intended.

To always force processing the whole end macro independently of this behaviour it is thus advisable to insert something that starts an empty partially filled line (\c) whenever there is a chance that a page break can happen. In the above example, the call of the ne request assures that the remaining code stays on the same page, so we have to insert \c only once.

The next example shows how to append three lines, then starting a new page unconditionally. Since '.ne 1' doesn't give the desired effect—there is always one line available or we are already at the beginning of the next page—we temporarily increase the page length by one line so that we can use '.ne 2'.

```
.de EM
.pl +1v
\c
.ne 2
line one
.br
\c
.ne 2
line two
.br
\c
.ne 2
line three
.br
.pl -1v
\c
'bp
..
.em EM
```

This specific feature affects only the first potential page break caused by the end macro; further page breaks emitted by the end macro are handled normally.

Another possible use of the em request is to make gtroff emit a single large page instead of multiple pages. For example, one may want to produce a long plain-text file for reading on-screen. The idea is to set the page length at the beginning of the document to a very large value to hold all the text, and automatically adjust it to the exact height of the document after the text has been output.

```
.de adjust-page-length
.  br
.  pl \\n[nl]u \" \n[nl] holds the current vert. position
..
.
.de single-page-mode
.  pl 99999
.  em adjust-page-length
..
.
.\" activate the above code
.single-page-mode
```

Since only one end-of-input trap does exist and other macro packages may already use it, care must be taken not to break the mechanism. A simple solution would be to append the above macro to the macro package's end-of-input macro using the `am` request.

5.25 Diversions

In `gtroff` it is possible to *divert* text into a named storage area. Due to the similarity to defining macros it is sometimes said to be stored in a macro. This is used for saving text for output at a later time, which is useful for keeping blocks of text on the same page, footnotes, tables of contents, and indices.

For orthogonality it is said that `gtroff` is in the *top-level diversion* if no diversion is active (i.e., the data is diverted to the output device).

Although the following requests can be used to create diversions, simply using an undefined diversion will cause it to be defined as empty. See Section 5.4 [Identifiers], page 69.

`.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.

The current partially filled line is included into the diversion. See the `box` request below for an example. Switching to another (empty) environment (with the `ev` request) avoids the inclusion of the current partially filled line.

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

Begin (or append to) a diversion like the `di` and `da` requests. The difference is that `box` and `boxa` do not include a partially filled line in the diversion.

Compare this:

```
Before the box.
.box xxx
In the box.
.br
.box
After the box.
.br
    ⇒ Before the box.  After the box.
.xxx
    ⇒ In the box.
```

with this:

```
Before the diversion.
.di yyy
In the diversion.
.br
.di
After the diversion.
.br
    ⇒ After the diversion.
.yyy
    ⇒ Before the diversion.  In the diversion.
```

`box` or `boxa` without an argument ends the diversion.

`\n[.z]` [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 or in the current diversion. It corresponds to the text baseline of the lowest line on the page. This is a read-only register.

```
.tm .h==\n[.h], nl==\n[nl]
    ⇒ .h==0, nl==-1
This is a test.
.br
.sp 2
.tm .h==\n[.h], nl==\n[nl]
    ⇒ .h==40, nl==120
```

As the previous example shows, 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. 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 that have been already stored in a macro are not taken into account.

```
.\" Center text both horizontally and 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 @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 that shouldn't be invoked until the diverted text is actually output.

The \! escape transparently embeds text up to and including the end of the line. The \? escape transparently embeds text until the next occurrence of the \? escape. Example:

```
\?anything\?
```

anything may not contain newlines; use \! to embed newlines in a diversion. The escape sequence \? is also recognized in copy mode and turned into a single internal code; it is this code that terminates *anything*. Thus the following example prints 4.

```
.nr x 1
.nf
.di d
\?\\?\\\\?\\\\\\\\nx\\\\?\\?\?
.di
.nr x 2
.di e
.d
.di
.nr x 3
.di f
.e
.di
.nr x 4
.f
```

Both escapes read the data in copy mode.

If \! is used in the top-level diversion, its argument is directly embedded into the gtroff intermediate output. This can be used for example to control a postprocessor that processes the data before it is sent to the device driver.

The \? escape used in the top-level diversion produces no output at all; its argument is simply ignored.

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

`.asciify` *div* [Request]

Unformat the diversion *div* in a way such that Unicode basic Latin (ASCII) characters, characters translated with the trin request, space

characters, and some escape sequences, that were formatted and diverted into *div* are treated like ordinary input characters when *div* is reread. Doing so can be useful in conjunction with the `writem` request. `asciify` 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
```

`asciify` cannot return all items in a diversion back to their source equivalent; nodes such as those produced by `\N[...]` will remain nodes, so the result cannot be guaranteed to be a pure string.

See Section 5.21.1 [Copy Mode], page 151.

`.unformat` *div* [Request]

Like `asciify`, unformat the diversion *div*. However, `unformat` handles only tabs and spaces between words, the latter usually arising from spaces or newlines in the input. Tabs are treated as input tokens, and spaces become stretchable again.

The vertical sizes of lines are not preserved, but glyph information (font, font size, space width, etc.) is retained. `unformat` can be useful in conjunction with the `box` and `boxa` requests.

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 69.) 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.

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 249. These four registers mark the top left and bottom right hand corners of a box that encompasses all written glyphs.

For example the input text:

```
Hello \O[0]world \O[1]this is a test.
```

produces the following output:

```
Hello       this is a test.
```

'`\O2`' Provided that the escape occurs at the outer level, enable output of glyphs and also write out to `stderr` the page number and four registers encompassing the glyphs previously written since the last call to `\O`.

'\O3' Begin a nesting level. At start-up, gtroff is at outer level. The current level is contained within the read-only register .O. See Section 5.6.5 [Built-in Registers], page 81.

'\O4' End a nesting level. The current level is contained within the read-only register .O. See Section 5.6.5 [Built-in Registers], page 81.

'\O[5Pfilename]'
This escape is grohtml specific. Provided that this escape occurs at the outer nesting level write the filename to stderr. The position of the image, *P*, must be specified and must be one of l, r, c, or i (left, right, centered, inline). *filename* is associated with the production of the next inline image.

5.28 Colors

.color [*n*] [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 174).

`\m` doesn't produce an input token in GNU `troff`. 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 174).

`\M` doesn't produce an input token in GNU `troff`.

5.29 I/O

`gtroff` has several requests for including files:

`.so` *file* [Request]

Read in the specified *file* and include 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 6.8 [gsoelim], page 199, 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 include 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 `-m` command-line option. If the file name to be included has the form *`name`*`.tmac` and it isn't found, `mso` tries to include `tmac.`*`name`* and vice versa. If the file does not exist, a warning of type '`file`' is emitted. See Section 5.33 [Debugging], page 188, for information about warnings.

`.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

```
.ev 1
.di x
.trf f
.di
.ev
```

The calls to `ev` prevent that the current partial input line becomes part of the diversion.

Both `trf` and `cf`, when used in a diversion, embeds an object in the diversion which, when reread, causes the contents of *file* to be transpar-

ently 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 69).

For `cf`, within a diversion, 'completely unprocessed' means that each line of a file to be inserted is handled as if it were preceded by `\!\\!`.

Both requests cause a line break.

`.nx` [*file*] [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. 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 GNU

troff to stop processing. If this were not there, troff 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'.

The intermediate output format of GNU troff 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 anywhere, so it is up to the user to do so.

This request causes an error if used in safer mode (which is the default). Use groff's or troff's -U option to activate unsafe mode.

For example, the following code fragment introduces the current time into a document:

```
.sy perl -e 'printf ".nr H %d\\n.nr M %d\\n.nr S %d\\n",\
             (localtime(time))[2,1,0]' > /tmp/x\n[$$]
.so /tmp/x\n[$$]
.sy rm /tmp/x\n[$$]
\nH:\nM:\nS
```

This works by having the Perl script (run by `sy`) print out the `nr` requests that set the number registers `H`, `M`, and `S`, and then reading 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 GNU `troff`, should be sufficient. Use the `af` request to format their values for 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 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 mode.

5.30 Postprocessor Access

There are two escapes that give information directly to the postprocessor. This is particularly useful for embedding PostScript into the final document.

`.device` *xxx* [Request]
\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.

Contrary to `\X`, the `device` request simply processes its argument in copy mode (see Section 5.21.1 [Copy Mode], page 151).

If the '`use_charnames_in_special`' keyword is set in the `DESC` file, special characters no longer cause an error; they are simply output verbatim. Additionally, the backslash is represented as `\\`.

'`use_charnames_in_special`' is currently used by `grohtml` only.

`.devicem` *xx* [Request]
\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 217).

See Chapter 7 [Output Devices], page 201.

5.31 Miscellaneous

This section documents parts of `gtroff` that 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 that are multiples of 5; this defaults to 1. *space* is the space to be left between the number and the text; this defaults to one digit space. The fourth argument is the indentation of the line numbers, defaulting to zero. Both *space* and *indent* are given as multiples of digit spaces; they can be negative also. Without any arguments, line numbers are turned off.

`gtroff` reserves three digit spaces for the line number (which is printed right-justified) plus the amount given by *indent*; the output lines are concatenated to the line numbers, separated by *space*, and *without* reducing the line length. Depending on the value of the horizontal page offset (as set with the `po` request), line numbers that are longer than the reserved space stick out to the left, or the whole line is moved to the right.

Parameters corresponding to missing arguments are not changed; any non-digit argument (to be more precise, any argument starting with a character valid as a delimiter for identifiers) is also treated as missing.

If line numbering has been disabled with a call to `nm` without an argument, it can be reactivated with '`.nm +0`', using the previously active line numbering parameters.

The parameters of `nm` are associated with the current environment (see Section 5.26 [Environments], page 174). 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.
```

The result is as follows.

```
 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.[38] The first argument is the glyph to be printed. The second argument is the distance away from the right margin. If missing, the previously set value is used; default is 10 pt). For text lines that are too long (that is, longer than the text length plus *dist*), the margin character is directly appended to the lines.

With no arguments the margin character is turned off. If this occurs before a break, no margin character is printed.

For compatibility with AT&T `troff`, a call to `mc` to set the margin character can't be undone immediately; at least one line gets a margin character. Thus

```
.ll 1i
.mc \[br]
.mc
xxx
.br
xxx
```

produces

```
xxx         |
xxx
```

For empty lines and lines produced by the `tl` request no margin character is emitted.

The margin character is associated with the current environment (see Section 5.26 [Environments], page 174).

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).

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

```
.ll 3i
.mc |
This paragraph is highlighted with a margin
character.
.sp
Vertical space isn't marked.
.br
\&
.br
But we can fake it with '\&'.
```

Result:

```
This  paragraph is highlighted |
with a margin character.       |

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*.[39] 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 69, for more on this topic.

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

For example, the input string 'fi\[:u]' is converted into a character token 'f', a character token 'i', and a special token ':u' (representing u umlaut). Later on, the character tokens 'f' and 'i' are merged to a single output node representing the ligature glyph 'fi' (provided the current font has a glyph for this ligature); the same happens with ':u'. All output glyph nodes are 'processed', which means that they are invariably associated with a given font, font size, advance width, etc. During the formatting process, gtroff itself adds various nodes to control the data flow.

Macros, diversions, and strings collect elements in two chained lists: a list of input tokens that have been passed unprocessed, and a list of output nodes. Consider the following the diversion.

```
.di xxx
a
\!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
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 that isn't stretchable anymore. To convert horizontal space nodes back to input tokens, use the unformat request.

Macros only contain elements in the token list (and the node list is empty); diversions and strings can contain elements in both lists.

Note that the chop request simply reduces the number of elements in a macro, string, or diversion by one. Exceptions are *compatibility save* and *compatibility ignore* input tokens, which are ignored. The substring request also ignores those input tokens.

Some requests like tr or cflags work on glyph identifiers only; this means that the associated glyph can be changed without destroying this

association. This can be very helpful for substituting glyphs. In the following example, we assume that glyph '`foo`' isn't available by default, so we provide a substitution using the `fchar` request and map it to input character '`x`'.

```
.fchar \[foo] foo
.tr x \[foo]
```

Now let us assume that we install an additional special font '`bar`' that has glyph '`foo`'.

```
.special bar
.rchar \[foo]
```

Since glyphs defined with `fchar` are searched before glyphs in special fonts, we must call `rchar` to remove the definition of the fallback glyph. Anyway, the translation is still active; '`x`' now maps to the real glyph '`foo`'.

Macro and request arguments preserve the compatibility mode:

```
.cp 1     \" switch to compatibility mode
.de xx
\\$1
..
.cp 0     \" switch compatibility mode off
.xx caf\['e]
    ⇒ café
```

Since compatibility mode is on while `de` is called, the macro `xx` activates compatibility mode while executing. Argument `$1` can still be handled properly because it inherits the compatibility mode status which was active at the point where `xx` is called.

After expansion of the parameters, the compatibility save and restore tokens are removed.

5.33 Debugging

`gtroff` is not easy to debug, but there are some useful features and strategies for debugging.

`.lf` *line* [*filename*] [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 that are split into many files, then put together with `soelim` and other preprocessors. Usually, it isn't invoked manually.

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]
Write *string* to the standard error stream (like `tm`)and then abort GNU `troff`; that is, stop processing and terminate with a failure status. With no argument, the message written is '`User Abort.`'.

`.ex` [Request]
Exit GNU `troff`; that is, stop processing and terminate with a successful status. To stop processing only the current file, use the `nx` request; See Section 5.29 [I/O], page 178.

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 such statements, use the `-r` option to set the register.

```
groff -rDB=1 file
```

If it is known in advance that there are many errors and no useful output, GNU `troff` can be forced to suppress formatted output with the `-z` option.

`.pev` [Request]
Report the contents of the current environment and all the currently defined environments (both named and numbered) to the standard error stream.

`.pm` [Request]
Report, to the standard error stream, the names of all defined macros, strings, and diversions with their sizes in bytes. Since GNU `troff` sometimes adds nodes by itself, the returned sizes can be larger than expected.

`.pnr` [Request]
Report the names and contents of all currently defined number registers to the standard error stream.

`.ptr` [Request]
Report the names and positions of all traps (not including input line traps and diversion traps) to the standard error stream. 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:

```
gtroff: backtrace: 'test':2: macro 'xxx'
gtroff: backtrace: 'test':5: macro 'yyy'
gtroff: backtrace: file 'test':8
```

The option `-b` of `gtroff` causes a backtrace to be generated on each error and warning. Warnings have to be enabled; see Section 5.33.1 [Warnings], page 191.

`\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]
Emit a `break` warning if the additional space inserted for each space between words in an output line adjusted to both margins with '`.ad b`' is larger than or equal to *limit*. A negative value is treated as zero; an absent argument toggles the warning on and off without changing *limit*. The default scaling indicator is '`m`'. At startup, `spreadwarn` is inactive and *limit* is 3 m.

For example,

```
.spreadwarn 0.2m
```

causes a warning if `break` warnings are not suppressed and `gtroff` must add 0.2 m or more for each interword space in a line. See Section 5.33.1 [Warnings], page 191.

`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.[40] This is enabled by default.

'`number`'
'`2`' Invalid numeric expressions. This is enabled by default. See Section 5.3 [Expressions], page 67.

'`break`'
'`4`' In fill mode, lines that could not be broken so that their length was less than the line length. This is enabled by default.

'`delim`'
'`8`' Missing or mismatched closing delimiters.

'`el`'
'`16`' Use of the `el` request with no matching `ie` request. See Section 5.20.3 [if-else], page 146.

'`scale`'
'`32`' Meaningless scaling indicators.

[40] `char` is a misnomer since it reports missing glyphs—there aren't missing input characters, only invalid ones.

‘`range`’
‘`64`’ Out of range arguments.

‘`syntax`’
‘`128`’ Invalid syntax.

‘`di`’
‘`256`’ Use of `di` or `da` without an argument when there is no current diversion.

‘`mac`’
‘`512`’ Use of undefined strings, macros and diversions. When an undefined string, macro, or diversion is used, that string is automatically defined as empty. So, in most cases, at most one warning is given for each name.

‘`reg`’
‘`1024`’ Use of undefined number registers. When an undefined number register is used, that register is automatically defined to have a value of 0. So, in most cases, at most one warning is given for use of a particular name.

‘`tab`’
‘`2048`’ Use of a tab character where a number was expected.

‘`right-brace`’
‘`4096`’ Use of `\}` where a number was expected.

‘`missing`’
‘`8192`’ Requests that are missing non-optional arguments.

‘`input`’
‘`16384`’ Invalid input characters.

‘`escape`’
‘`32768`’ Unrecognized escape sequences. When an unrecognized escape sequence `\`*X* is encountered, the escape character is ignored, and *X* is printed.

‘`space`’
‘`65536`’ Missing space between a request or macro and its argument. This warning is given when an undefined name longer than two characters is encountered, and the first two characters of the name make a defined name. The request or macro is not invoked. When this warning is given, no macro is automatically defined. This is enabled by default. This warning never occurs in compatibility mode.

‘`font`’
‘`131072`’ Non-existent fonts. This is enabled by default.

‘ig’
‘262144’ Invalid escapes in text ignored with the ig request. These are conditions that are errors when they do not occur in ignored text.

‘color’
‘524288’ Color related warnings.

‘file’
‘1048576’ Missing files. The mso request gives this warning when the requested macro file does not exist. This is enabled by default.

‘all’ All warnings except ‘di’, ‘mac’ and ‘reg’. It is intended that this covers all warnings that are useful with traditional macro packages.

‘w’ All warnings.

5.34 Implementation Differences

GNU troff has a number of features that cause incompatibilities with documents written using old versions of troff. Some GNU extensions to troff have become supported by other implementations.

GNU troff does not always hyphenate words as AT&T troff does. The AT&T implementation uses a set of hard-coded rules specific to U.S. English, while GNU troff uses language-specific hyphenation pattern files derived from TEX. Furthermore, in old versions of troff there was a limited amount of space to store hyphenation exceptions (arguments to the hw request); GNU troff has no such restriction.

Long names may be GNU troff’s most obvious innovation. AT&T troff interprets ‘.dsabcd’ as defining a string ‘ab’ with contents ‘cd’. Normally, GNU troff interprets this as a call of a macro named dsabcd. AT&T troff also interprets *[and \n[as a reference to a string or number register, respectively, called ‘[’. In GNU troff, however, the ‘[’ is normally interpreted as delimiting a long name. In compatibility mode, GNU troff interprets names in the traditional way, which means that they are limited to one or two characters.

.cp [*n*] [Request]
.do *name* [Request]
\n[.C] [Register]
\n[.cp] [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 interprets the string, request, diversion, or macro *name* (along with any further arguments) with compatibility mode disabled. Compatibility mode is restored (only if it was active) when the *expansion* of *name* is interpreted; that is, the restored compatibility state applies to the contents of the macro (string, ...) *name* as well as file or pipe data read if *name* is the `so`, `mso`, or `pso` request.

The following example illustrates several aspects of `do` behavior.

```
.de mac1
FOO
..
.de1 mac2
groff
.mac1
..
.de mac3
compatibility
.mac1
..
.de ma
\\$1
..
.cp 1
.do mac1
.do mac2 \" mac2, defined with .de1, calls "mac1"
.do mac3 \" mac3 calls "ma" with argument "c1"
.do mac3 \[ti] \" groff syntax accepted in .do arguments
    ⇒ FOO groff FOO compatibility c1 ~
```

The read-only number register `.cp`, meaningful only when dereferenced from a `do` request, is 1 if compatibility mode was on when the `do` request was encountered, and 0 if it was not. This register is specialized and may require a statement of rationale.

When writing macro packages or documents that use GNU `troff` features and which may be mixed with other packages or documents that do not—common scenarios include serial processing of man pages or use of the `so` or `mso` requests—you may desire correct operation regardless of compatibility mode in the surrounding context. It may occur to you to save the existing value of '`\n(.C`' into a register, say, '`_C`', at the beginning of your file, turn compatibility mode off with '`.cp 0`', then restore it from that register at the end with '`.cp \n(_C`'. At the same time, a modular design of a document or macro package may lead you to multiple layers of inclusion. You cannot use the same register name everywhere or you risk "clobbering" the value from a preceding or enclosing context. The two-character register name space of AT&T `troff` is confining and mnemonically challenging; you may wish to use the more capacious name space of GNU `troff`. However, attempting '`.nr _my_saved_C \n(.C`'

will not work in compatibility mode; the register name is too long. "This is exactly what do is for," you think, '.do nr _my_saved_C \n(.C'. The foregoing will always save zero to your register, because do turns compatibility mode *off* while it interprets its argument list. What you need is:

```
.do nr _my_saved_C \n[.cp]
.cp 0
```

at the beginning of your file, followed by

```
.cp _my_saved_C
```

at the end. As in the C language, we all have to share one big name space, so choose a register name that is unlikely to collide with other uses.

Normally, GNU troff preserves the input level in delimited arguments, but not in compatibility mode.

```
.ds xx '
\w'abc\*(xxdef'
```

⇒ 168 (normal mode on a terminal device)
⇒ 72def' (compatibility mode on a terminal device)

Furthermore, the escapes \f, \H, \m, \M, \R, \s, and \S are transparent for recognizing the beginning of a line only in compatibility mode. For example, this code produces bold output in both cases, but the text differs.

```
.de xx
Hello!
..
\fB.xx\fP
```

⇒ .xx (normal mode)
⇒ Hello! (compatibility mode)

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; AT&T troff does. The \A escape sequence (see Section 5.4 [Identifiers], page 69) may be helpful in avoiding use of these escape sequences in names.

Normally, the syntax form \s*n* accepts only a single character (a digit) for *n*, consistently with other forms that originated in AT&T troff, like *, \$, \f, \g, \k, \n, and \z. In compatibility mode only, a non-zero *n* must be in the range 4–39. Legacy documents relying upon this quirk of parsing[41] should be migrated to another \s form.

Fractional point sizes cause one noteworthy incompatibility. In AT&T troff the ps request ignores scale indicators and thus '.ps 10u' sets the

[41] The Graphic Systems C/A/T phototypesetter (the original device target for AT&T troff) supported only a few discrete point sizes in the range 6–36, so Ossanna contrived a special case in the parser to do what the user must have meant. Kernighan warned of this in the 1992 revision of CSTR #54 (§2.3), and more recently, McIlroy referred to it as a "living fossil".

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 136.

The `pm` request differs from AT&T `troff`: GNU `troff` reports the sizes of macros, strings, and diversions in bytes and ignores an argument to report only the sum of the sizes.

Unlike AT&T `troff`, GNU `troff` does not ignore the `ss` request if the output is a terminal device; instead, the values of minimal inter-word and additional inter-sentence spacing are each rounded down to the nearest multiple of 12.

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 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. Special characters can be both: before being added to the output, they act as input entities; afterwards, they denote glyphs. 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. Consider the following example.

```
.di x
\\\\
.br
.di
.x
```

It 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. AT&T `troff` would interpret them as escape characters when they were reread and would end up printing one '\'.

One correct way to obtain a printable backslash in most documents is to use the `\e` escape sequence; this always prints a single instance of the current escape character[42], regardless of whether or not it is used in a diversion; it also works in both GNU `troff` and AT&T `troff`.

The other correct way, appropriate in contexts independent of the backslash's common use as a `troff` escape character—perhaps in discussion of character sets or other programming languages—is the character escape `\(rs`

[42] Naturally, if you've changed the escape character, you need to prefix the `e` with whatever it is—and you'll likely get something other than a backslash in the output.

or \[rs], for "reverse solidus", from its name in the ECMA-6 (ISO/IEC 646) standard[43].

To store an escape sequence in a diversion that is interpreted when the diversion is reread, either use the traditional \! transparent output facility, or, if this is unsuitable, the new \? escape sequence. See Section 5.25 [Diversions], page 170, and Section 5.32 [Gtroff Internals], page 186.

[43] This character escape is not portable to AT&T troff, but is to its lineal descendant, Heirloom Doctools troff, as of its 060716 release (July 2006).

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 `gchem`

6.6.1 Invoking `gchem`

6.7 `grefer`

6.7.1 Invoking `grefer`

6.8 `gsoelim`

6.8.1 Invoking `gsoelim`

6.9 `preconv`

6.9.1 Invoking `preconv`

7 Output Devices

7.1 Special Characters

See Section 8.2 [Device and Font Files], page 222.

7.2 grotty

The postprocessor `grotty` translates the output from GNU `troff` into a form suitable for typewriter-like devices. It is fully documented on its manual page, *grotty(1)*.

7.2.1 Invoking grotty

The postprocessor `grotty` accepts the following command-line options:

`-b` Do not overstrike bold glyphs. Ignored if `-c` isn't used.

`-B` Do not underline bold-italic glyphs. Ignored if `-c` isn't used.

`-c` Use overprint and disable colours for printing on legacy Teletype printers (see below).

`-d` Do not render lines (that is, ignore all `\D` escapes).

`-f` Use form feed control characters in the output.

`-F`*dir* Put the directory *dir*`/dev`*name* in front of the search path for the font and device description files, given the target device *name*.

`-h` Use horizontal tabs for sequences of 8 space characters.

`-i` Request italic glyphs from the terminal. Ignored if `-c` is active.

`-o` Do not overstrike.

`-r` Highlight italic glyphs. Ignored if `-c` is active.

`-u` Do not underline italic glyphs. Ignored if `-c` isn't used.

`-U` Do not overstrike bold-italic glyphs. Ignored if `-c` isn't used.

`-v` Print the version number.

The `-c` option tells `grotty` to use an output format compatible with paper terminals, like the Teletype machines for which `roff` and `nroff` were first developed but which are no longer in wide use. SGR escape sequences (from ISO 6429) are not emitted. Instead, `grotty` overstrikes, representing a bold character *c* with the sequence '*c* `BACKSPACE` *c*' and an italic character *c* with the sequence '`_ BACKSPACE` *c*'. Furthermore, color output is disabled. The same effect can be achieved either by setting the `GROFF_NO_SGR` environment variable or by using a `groff` escape sequence within the document; see the subsection "Device control commands" of the *grotty(1)* man page for details.

The legacy output format can be rendered on a video terminal (or emulator) by piping `grotty`'s output through `ul`, which may render bold italics as reverse video. Some implementations of `more` are also able to display these sequences; you may wish to experiment with that command's `-b` option. `less` renders legacy bold and italics without requiring options. In contrast to the teletype output drivers of some other `roff` implementations, `grotty` never outputs reverse line feeds. There is therefore no need to filter its output through `col`.

7.3 grops

The postprocessor `grops` translates the output from GNU `troff` into a form suitable for Adobe PostScript devices. It is fully documented on its manual page, *grops(1)*.

7.3.1 Invoking grops

The postprocessor `grops` accepts the following command-line options:

`-b`*flags* Use backward compatibility settings given by *flags* as documented in the *grops(1)* manual page. Overrides the command `broken` in the `DESC` file.

`-c`*n* Print *n* copies of each page.

`-F`*dir* Put the directory *dir*`/dev`*name* in front of the search path for the font, prologue and device description files, given the target device *name*, usually **ps**.

`-g` Tell the printer to guess the page length. Useful for printing vertically centered pages when the paper dimensions are determined at print time.

`-I`*path* `...`
Consider the directory *path* for searching included files specified with relative paths. The current directory is searched as fallback.

`-l` Use landscape orientation.

`-m` Use manual feed.

`-p`*papersize*
Set the page dimensions. Overrides the commands `papersize`, `paperlength`, and `paperwidth` in the `DESC` file. See the *groff_font(5)* manual page for details.

`-P`*prologue*
Use the *prologue* in the font path as the prologue instead of the default `prologue`. Overrides the environment variable `GROPS_PROLOGUE`.

`-wn` Set the line thickness to $n/1000$ em. Overrides the default value $n = 40$.

`-v` Print the version number.

7.3.2 Embedding PostScript

The escape sequence

‘`\X'ps: import` *file llx lly urx ury width* `[`*height*`]'`’

places a rectangle of the specified *width* containing the PostScript drawing from file *file* bound by the box from *llx lly* to *urx ury* (in PostScript coordinates) at the insertion point. If *height* is not specified, the embedded drawing is scaled proportionally.

See Section 5.31 [Miscellaneous], page 184, for the `psbb` request, which automatically generates the bounding box.

This escape sequence is used internally by the macro `PSPIC` (see the *groff_tmac(5)* manual page).

7.4 gropdf

The postprocessor `gropdf` translates the output from GNU `troff` into a form suitable for Adobe PDF devices. It is fully documented on its manual page, *gropdf(1)*.

7.4.1 Invoking gropdf

The postprocessor `gropdf` accepts the following command-line options:

`-d` Produce uncompressed PDFs that include debugging comments.

`-e` This forces `gropdf` to embed all used fonts in the PDF, even if they are one of the 14 base Adobe fonts.

`-Fdir` Put the directory *dir*`/dev`*name* in front of the search path for the font, prologue and device description files, given the target device *name*, usually **pdf**.

`-y`*foundry*
: This forces the use of a different font foundry.

`-l` Use landscape orientation.

`-p`*papersize*
: Set the page dimensions. Overrides the commands `papersize`, `paperlength`, and `paperwidth` in the `DESC` file. See the *groff_font(5)* manual page for details.

`-v` Print the version number.

`-s` Append a comment line to end of PDF showing statistics, i.e. number of pages in document. Ghostscript's *ps2pdf(1)* complains about this line if it is included, but works anyway.

`-u`*filename*

`gropdf` normally includes a ToUnicode CMap with any font created using `text.enc` as the encoding file, this makes it easier to search for words that contain ligatures. You can include your own CMap by specifying a *filename* or have no CMap at all by omitting the *filename*.

7.4.2 Embedding PDF

The escape sequence

‘`\X'pdf: pdfpic` *file alignment width* `[`*height*`] [`*linelength*`]'`’

places a rectangle of the specified *width* containing the PDF drawing from file *file* of desired *width* and *height* (if *height* is missing or zero then it is scaled proportionally). If *alignment* is `-L` the drawing is left aligned. If it is `-C` or `-R` a *linelength* greater than the width of the drawing is required as well. If *width* is specified as zero then the width is scaled in proportion to the height.

7.5 `grodvi`

The postprocessor `grodvi` translates the output from GNU `troff` into the DVI format produced by the TEX document preparation system. This enables the use of programs that process the DVI format, like `dvips` and `dvipdf`, with GNU `troff` output. `grodvi` is fully documented in its manual page, *grodvi(1)*.

7.5.1 Invoking `grodvi`

The postprocessor `grodvi` accepts the following command-line options:

`-d` Do not use **tpic** specials to implement drawing commands.

`-F`*dir* Put the directory *dir*`/dev`*name* in front of the search path for the font and device description files, given the target device *name*, usually **dvi**.

`-l` Use landscape orientation.

`-p`*papersize*

Set the page dimensions. Overrides the commands `papersize`, `paperlength`, and `paperwidth` in the `DESC` file. See *groff_font(5)* manual page for details.

`-v` Print the version number.

`-w`*n* Set the line thickness to $n/1000$ em. Overrides the default value $n = 40$.

7.6 grolj4

The postprocessor `grolj4` translates the output from GNU `troff` into the **PCL5** output format suitable for printing on a **HP LaserJet 4** printer. It is fully documented on its manual page, *grolj4(1)*.

7.6.1 Invoking grolj4

The postprocessor `grolj4` accepts the following command-line options:

`-cn` Print *n* copies of each page.

`-Fdir` Put the directory *dir*`/dev`*name* in front of the search path for the font and device description files, given the target device *name*, usually **lj4**.

`-l` Use landscape orientation.

`-p`*size* Set the page dimensions. Valid values for *size* are: `letter`, `legal`, `executive`, `a4`, `com10`, `monarch`, `c5`, `b5`, `d1`.

`-v` Print the version number.

`-wn` Set the line thickness to $n/1000$ em. Overrides the default value $n = 40$.

The special drawing command '`\D'R` *dh dv*`'` draws a horizontal rectangle from the current position to the position at offset (*dh*,*dv*).

7.7 grolbp

The postprocessor `grolbp` translates the output from GNU `troff` into the **LBP** output format suitable for printing on **Canon CAPSL** printers. It is fully documented on its manual page, *grolbp(1)*.

7.7.1 Invoking grolbp

The postprocessor `grolbp` accepts the following command-line options:

`-cn` Print *n* copies of each page.

`-Fdir` Put the directory *dir*`/dev`*name* in front of the search path for the font, prologue and device description files, given the target device *name*, usually **lbp**.

`-l` Use landscape orientation.

`-o`*orientation*
Use the *orientation* specified: `portrait` or `landscape`.

`-p`*papersize*
Set the page dimensions. See *groff_font(5)* manual page for details.

`-wn` Set the line thickness to $n/1000$ em. Overrides the default value $n = 40$.

`-v` Print the version number.

`-h` Print command-line help.

7.8 grohtml

The `grohtml` front end (which consists of a preprocessor, `pre-grohtml`, and a device driver, `post-grohtml`) translates the output of GNU `troff` to HTML. Users should always invoke `grohtml` via the `groff` command with a `\-Thtml` option. If no files are given, `grohtml` will read the standard input. A filename of `-` will also cause `grohtml` to read the standard input. HTML output is written to the standard output. When `grohtml` is run by `groff`, options can be passed to `grohtml` using `groff`'s `-P` option.

`grohtml` invokes `groff` twice. In the first pass, pictures, equations, and tables are rendered using the `ps` device, and in the second pass HTML output is generated by the `html` device.

`grohtml` always writes output in `UTF-8` encoding and has built-in entities for all non-composite Unicode characters. In spite of this, `groff` may issue warnings about unknown special characters if they can't be found during the first pass. Such warnings can be safely ignored unless the special characters appear inside a table or equation, in which case glyphs for these characters must be defined for the `ps` device as well.

This output device is fully documented on its manual page, *grohtml(1)*.

7.8.1 Invoking grohtml

The postprocessor `grohtml` accepts the following command-line options:

`-abits` Use this number of *bits* (= 1, 2 or 4) for text antialiasing. Default: *bits* = 4.

`-a0` Do not use text antialiasing.

`-b` Use white background.

`-Ddir` Store rendered images in the directory *dir*.

`-Fdir` Put the directory *dir*`/dev`*name* in front of the search path for the font, prologue and device description files, given the target device *name*, usually **html**.

`-gbits` Use this number of *bits* (= 1, 2 or 4) for antialiasing of drawings. Default: *bits* = 4.

`-g0` Do not use antialiasing for drawings.

`-h` Use the `B` element for section headings.

`-i`*resolution*
: Use the *resolution* for rendered images. Default: *resolution* = 100 dpi.

`-I`*stem*
: Set the images' *stem name*. Default: *stem* = `grohtml-`*XXX* (*XXX* is the process ID).

`-j`*stem*
: Place each section in a separate file called *stem*`-`*n*`.html` (where *n* is a generated section number).

`-l`
: Do not generate the table of contents.

`-n`
: Generate simple fragment identifiers.

`-o`*offset*
: Use vertical padding *offset* for images.

`-p`
: Display the page rendering progress to `stderr`.

`-r`
: Do not use horizontal rules to separate headers and footers.

`-s`*size*
: Set the base font size, to be modified using the elements `BIG` and `SMALL`.

`-S`*level*
: Generate separate files for sections at level *level*.

`-v`
: Print the version number.

`-V`
: Generate a validator button at the bottom.

`-y`
: Generate a signature of groff after the validator button, if any.

7.8.2 `grohtml` specific registers and strings

`\n[ps4html]` [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
```

or

```
troff -Txhtml
```

The PostScript device is used to create all the image files (for `-Thtml`; if `-Txhtml` is used, all equations are passed to `geqn` to produce MathML, and the register `ps4html` enables the macro sets to ignore floating keeps, footers, and headings.

The register `www-image-template` is set to the user specified template name or the default name.

7.9 gxditview

7.9.1 Invoking gxditview

8 File formats

All files read and written by `gtroff` are text files. The following two sections describe their format.

8.1 `gtroff` Output

This section describes the intermediate output format of GNU `troff`. This output is produced by a run of `gtroff` before it is fed into a device postprocessor program.

As `groff` is a wrapper program around `gtroff` that automatically calls a postprocessor, this output does not show up normally. This is why it is called *intermediate*. `groff` provides the option `-Z` to inhibit postprocessing, such that the produced intermediate output is sent to standard output just like calling `gtroff` manually.

Here, the term *troff output* describes what is output by `gtroff`, while *intermediate output* refers to the language that is accepted by the parser that prepares this output for the postprocessors. This parser is more tolerant of 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.

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

8.1.1.1 Separation

AT&T `troff` output has strange requirements regarding whitespace. The `gtroff` output parser, however, is more tolerant, making whitespace maximally optional. Such characters, i.e., the tab, space, and newline, 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 stacking of such commands on the same line, but fortunately, in `gtroff`'s intermediate output, every command with at least one argument is followed by a line break, thus providing excellent readability.

The other commands—those for drawing and device controlling—have a more complicated structure; some recognize long command names, and some take a variable number of arguments. So all '`D`' and '`x`' commands were designed to request a syntactical line break after their last argument. Only one command, '`x X`', has an argument that can stretch over several lines; all other commands must have all of their arguments on the same line as the command, i.e., the arguments may not be split by a line break.

Empty lines (these are lines containing only space and/or a comment), can occur everywhere. They are just ignored.

8.1.1.2 Argument Units

Some commands take integer arguments that are assumed to represent values in a measurement unit, but the letter for the corresponding scale indicator is not written with the output command arguments. Most commands assume the scale indicator '`u`', the basic unit of the device, some use '`z`', the scaled point unit of the device, while others, such as the color commands, expect plain integers.

Single characters can have the eighth bit set, as can the names of fonts and special characters. The names of characters and fonts can be of arbitrary length. A character that is to be printed is always in the current font.

A string argument is always terminated by the next whitespace character (space, tab, or newline); an embedded '`#`' character is regarded as part of

the argument, not as the beginning of a comment command. An integer argument is already terminated by the next non-digit character, which then is regarded as the first character of the next argument or command.

8.1.1.3 Document Parts

A correct intermediate output document consists of two parts, the *prologue* and the *body*.

The task of the prologue is to set the general device parameters using three exactly specified commands. `gtroff`'s prologue is guaranteed to consist of the following three lines (in that order):

```
x T device
x res n h v
x init
```

with the arguments set as outlined in Section 8.1.2.4 [Device Control Commands], page 217. 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 tolerant of 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 analogous command for the filling color of graphic objects is '`DF`'. The color components are specified as integer arguments between 0 and 65536. The number of color components and their meaning vary for the different color schemes. These commands are generated by `gtroff`'s escape sequence `\m`. No position changing. These commands are a `gtroff` extension.

  `mc` *cyan magenta yellow*
  : Set color using the CMY color scheme, having the 3 color components *cyan*, *magenta*, and *yellow*.

  `md`
  : Set color to the default color value (black in most cases). No component arguments.

  `mg` *gray*
  : Set color to the shade of gray given by the argument, an integer between 0 (black) and 65536 (white).

  `mk` *cyan magenta yellow black*
  : Set color using the CMYK color scheme, having the 4 color components *cyan*, *magenta*, *yellow*, and *black*.

[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 221.

`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 222).

`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 222).

`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 159.

Unknown '`D`' commands are assumed to be device-specific. Its arguments are parsed as strings; the whole information is then sent to the postprocessor.

In the following command reference, the syntax element ⟨line break⟩ means a syntactical line break as defined above.

`D~` *h1 v1 h2 v2 ... hn vn*⟨line break⟩
Draw B-spline from current position to offset (*h1*,*v1*), then to offset (*h2*,*v2*), if given, etc. up to (*hn*,*vn*). This command takes a variable number of argument pairs; the current position is moved to the terminal point of the drawn curve.

`Da` *h1 v1 h2 v2*⟨line break⟩
Draw arc from current position to (*h1*,*v1*)+(*h2*,*v2*) with center at (*h1*,*v1*); then move the current position to the final point of the arc.

`DC` *d*⟨line break⟩
`DC` *d dummy-arg*⟨line break⟩
: Draw a solid circle using the current fill color with diameter *d* (integer in basic units '`u`') with leftmost point at the current position; then move the current position to the rightmost point of the circle. An optional second integer argument is ignored (this allows the formatter to generate an even number of arguments). This command is a `gtroff` extension.

`Dc` *d*⟨line break⟩
: Draw circle line with diameter *d* (integer in basic units '`u`') with leftmost point at the current position; then move the current position to the rightmost point of the circle.

`DE` *h v*⟨line break⟩
: Draw a solid ellipse in the current fill color with a horizontal diameter of *h* and a vertical diameter of *v* (both integers in basic units '`u`') with the leftmost point at the current position; then move to the rightmost point of the ellipse. This command is a `gtroff` extension.

`De` *h v*⟨line break⟩
: Draw an outlined ellipse with a horizontal diameter of *h* and a vertical diameter of *v* (both integers in basic units '`u`') with the leftmost point at current position; then move to the rightmost point of the ellipse.

`DF` *color-scheme* [*component ...*]⟨line break⟩
: Set fill color for solid drawing objects using different color schemes; the analogous command for setting the color of text, line graphics, and the outline of graphic objects is '`m`'. The color components are specified as integer arguments between 0 and 65536. The number of color components and their meaning vary for the different color schemes. These commands are generated by `gtroff`'s escape sequences '`\D'F ...'`' and `\M` (with no other corresponding graphics commands). No position changing. This command is a `gtroff` extension.

  `DFc` *cyan magenta yellow*⟨line break⟩
  : Set fill color for solid drawing objects using the CMY color scheme, having the 3 color components *cyan*, *magenta*, and *yellow*.

  `DFd`⟨line break⟩
  : Set fill color for solid drawing objects to the default fill color value (black in most cases). No component arguments.

`DFg` *gray*⟨line break⟩
: Set fill color for solid drawing objects to the shade of gray given by the argument, an integer between 0 (black) and 65536 (white).

`DFk` *cyan magenta yellow black*⟨line break⟩
: Set fill color for solid drawing objects using the CMYK color scheme, having the 4 color components *cyan*, *magenta*, *yellow*, and *black*.

`DFr` *red green blue*⟨line break⟩
: Set fill color for solid drawing objects using the RGB color scheme, having the 3 color components *red*, *green*, and *blue*.

`Df` *n*⟨line break⟩
: The argument *n* must be an integer in the range −32767 to 32767.

  $0 \le n \le 1000$
  : 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 210).

`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 118.

`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 221.

`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 GNU `troff`, this is 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 GNU `troff`, arbitrary syntactical space around and within this command is allowed. Only when a preceding command on the same line ends with an argument of variable length is a separating space 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 much lower resolution than modern printers, the intermediate output for X11 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-independent TROFF*, written by Brian Kernighan.

The `gtroff` intermediate output format is compatible with this specification except for the following features.

- The classical quasi-device independence is not yet implemented.
- The old hardware was very different from what we use today. So the `groff` devices are also fundamentally different from the ones in AT&T `troff`. For example, the AT&T PostScript device is called `post` and has a resolution of only 720 units per inch, suitable for printers 20 years ago, while `groff`'s `ps` device has a resolution of 72000 units per

inch. Maybe, by implementing some rescaling mechanism similar to the classical quasi-device independence, `groff` could emulate AT&T's `post` device.

- The B-spline command '`D~`' is correctly handled by the intermediate output parser, but the drawing routines aren't implemented in some of the postprocessor programs.
- The argument of the commands '`s`' and '`x H`' has the implicit unit scaled point '`z`' in `gtroff`, while AT&T `troff` has point ('`p`'). This isn't an incompatibility but a compatible extension, for both units coincide for all devices without a `sizescale` parameter in the `DESC` file, including all postprocessors from AT&T and `groff`'s text devices. The few `groff` devices with a `sizescale` parameter either do not exist for AT&T `troff`, have a different name, or seem to have a different resolution. So conflicts are very unlikely.
- The position changing after the commands '`Dp`', '`DP`', and '`Dt`' is illogical, but as old versions of `gtroff` used this feature it is kept for compatibility reasons.

8.2 Device and Font Files

The GNU `troff` font format is a rough superset of the AT&T device-independent `troff` font format. In distinction to the AT&T implementation, GNU `troff` lacks a binary format; all files are text files.[3] The font files for device *name* are stored in a directory `devname`. There are two types of file: a device description file called `DESC` and for each font *f* a font file called `f`.

8.2.1 `DESC` File Format

The `DESC` file can contain the following types of line. Except for the `charset` keyword, which must come last (if at all), the order of the lines is not important. Later entries in the file, however, override previous values.

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

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

`fonts` *n F1 F2 F3 ... Fn*
Fonts *F1* ... *Fn* are mounted in the font positions *m*+1, ..., *m*+*n* where *m* is the number of styles. This command may extend over more than one line. A font name of 0 means no font is mounted on the corresponding font position.

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

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

`image_generator` *`string`*
: Needed for `grohtml` only. It specifies the program to generate PNG images from PostScript input. Under GNU/Linux this is usually `gs` but under other systems (notably cygwin) it might be set to another name.

`paperlength` *`n`*
: The physical vertical dimension of the output medium in machine units. This isn't used by `troff` itself but by output devices. Deprecated. Use `papersize` instead.

`papersize` *`string`* `...`
: Select a paper size. Valid values for *string* are the ISO paper types `A0`–`A7`, `B0`–`B7`, `C0`–`C7`, `D0`–`D7`, `DL`, and the US paper types `letter`, `legal`, `tabloid`, `ledger`, `statement`, `executive`, `com10`, and `monarch`. Case is not significant for *string* if it holds predefined paper types. Alternatively, *string* can be a file name (e.g. `/etc/papersize`); if the file can be opened, `groff` reads the first line and tests for the above paper sizes. Finally, *string* can be a custom paper size in the format *`length`*`,`*`width`* (no spaces before and after the comma). Both *length* and *width* must have a unit appended; valid values are '`i`' for inches, '`c`' for centimeters, '`p`' for points, and '`P`' for picas. Example: `12c,235p`. An argument that starts with a digit is always treated as a custom paper format. `papersize` sets both the vertical and horizontal dimension of the output medium.

  More than one argument can be specified; `groff` scans from left to right and uses the first valid paper specification.

`paperwidth` *`n`*
: The physical horizontal dimension of the output medium in machine units. This isn't used by `troff` itself but by output devices. Deprecated. Use `papersize` instead.

`pass_filenames`
: Tell `gtroff` to emit the name of the source file currently being processed. This is achieved by the intermediate output command '`F`'. Currently, this is only used by the `grohtml` output device.

`postpro` *`program`*
: Call *program* as a postprocessor. For example, the line

  ```
  postpro grodvi
  ```

  in the file `devdvi/DESC` makes `groff` call `grodvi` if option `-Tdvi` is given (and `-Z` isn't used).

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

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

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

`sizes` *s1 s2 ... sn* `0`
: This means that the device has fonts at *s1*, *s2*, ... *sn* scaled points. The list of sizes must be terminated by 0 (this is digit zero). Each *si* can also be a range of sizes *m*–*n*. The list can extend over more than one line.

`sizescale` *n*
: The scale factor for point sizes. By default this has a value of 1. One scaled point is equal to one point/*n*. The arguments to the `unitwidth` and `sizes` commands are given in scaled points. See Section 5.18.2 [Fractional Type Sizes], page 136.

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

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

`unicode`
: Indicate that the output device supports the complete Unicode repertoire. Useful only for devices that produce *character entities* instead of glyphs.

  If `unicode` is present, no `charset` section is required in the font description files since the Unicode handling built into `groff` is used. However, if there are entries in a `charset` section, they either override the default mappings for those particular characters or add new mappings (normally for composite characters).

  This is used for `-Tutf8`, `-Thtml`, and `-Txhtml`.

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

`unscaled_charwidths`
: Make the font handling module always return unscaled character widths. Needed for the `grohtml` device.

`use_charnames_in_special`
: This command indicates that `gtroff` should encode special characters inside special commands. Currently, this is only used by the `grohtml` output device. See Section 5.30 [Postprocessor Access], page 183.

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

The `res`, `unitwidth`, `fonts`, and `sizes` lines are mandatory. Other commands are ignored by `gtroff` but may be used by postprocessors to store arbitrary information about the device in the `DESC` file.

GNU `troff` recognizes but completely ignores the obsolete keywords `spare1`, `spare2`, and `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*]

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

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`'. Single-letter special characters can't be accessed as '`\c`'; the only exception is '`\-`', which is identical to `\[-]`.

`gtroff` supports 8-bit input characters; however some utilities have difficulties with eight-bit characters. For this reason, there is a convention that the entity name '`charn`' is equivalent to the single input character whose code is *n*. For example, '`char163`' would be equivalent to the character with code 163, which is the pounds sterling sign in the ISO Latin-1 character set. You shouldn't use '`charn`' entities in font description files since they are related to input, not output. Otherwise, you get hard-coded connections between input and output encoding, which prevents use of different (input) character sets.

The name '`---`' is special and indicates that the glyph is unnamed; such glyphs can only be used by means of the `\N` escape sequence in `gtroff`.

The *type* field gives the glyph type:

`1` the glyph has a descender, for example, '`p`';

`2` the glyph has an ascender, for example, '`b`';

`3` the glyph has both an ascender and a descender, for example, '`(`'.

The *code* field gives the code that the postprocessor uses to print the glyph. The glyph can also be input to `gtroff` using this code by means of the `\N` escape sequence. *code* can be any integer. If it starts with '`0`' it is interpreted as octal; if it starts with '`0x`' or '`0X`' it is interpreted as hexadecimal. Note, however, that the `\N` escape sequence only accepts a decimal integer.

The *entity-name* field gives an ASCII string identifying the glyph that the postprocessor uses to print the `gtroff` glyph *name*. This field is optional and has been introduced so that the `grohtml` device driver can encode its character set. For example, the glyph '`\[Po]`' is represented as '`&pound;`' in HTML 4.0.

Anything on the line after the *entity-name* field resp. after '`--`' is ignored.

The *metrics* field has the form:

width[`,`*height*[`,`*depth*[`,`*italic-correction*
[`,`*left-italic-correction*[`,`*subscript-correction*]]]]]

There must not be any spaces between these subfields (it has been split here into two lines for better legibility only). Missing subfields are assumed to

[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.

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 AT&T device-independent `troff`. 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

Version 1.3, 3 November 2008

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.

The "publisher" means any person or entity that distributes copies of the Document to the public.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

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 under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

   However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

   Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

   Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

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. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

11. RELICENSING

"Massive Multiauthor Collaboration Site" (or "MMC Site") means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A "Massive Multiauthor Collaboration" (or "MMC") contained in the site means any set of copyrightable works thus published on the MMC site.

"CC-BY-SA" means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

"Incorporate" means to publish or republish a Document, in whole or in part, as part of another Document.

An MMC is "eligible for relicensing" if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.

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 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled ‘‘GNU Free Documentation License’’.

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with. . . Texts." line with this:

> with the Invariant Sections being *list their titles*, with the Front-Cover Texts being *list*, and with the Back-Cover Texts being *list*.

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

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

T

U

V

W

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]`'.

$

%

.

C

D

F

G

H

L

M

N

O

P

Q

R

S

U

V

Y

F Macro Index

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

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]`'.

C

D

F

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

T

U

K Concept Index

|

8

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z