# groff

The GNU implementation of `troff`
Edition 1.23.0
February 2023

**Trent A. Fisher**
**Werner Lemberg**
**G. Branden Robinson**

This manual documents GNU `troff` version 1.23.0.

# Table of Contents

# 1 Introduction

GNU `roff` (or `groff`) is a programming system for typesetting documents. It is highly flexible and has been used extensively for over thirty years.

## 1.1 Background

M. Douglas McIlroy, formerly of AT&T Bell Laboratories and present at the creation of the Unix operating system, offers an authoritative historical summary.

> The prime reason for Unix was the desire of Ken [Thompson], Dennis [Ritchie], and Joe Ossanna to have a pleasant environment for software development. The fig leaf that got the nod from ... management was that an early use would be to develop a "stand-alone" word-processing system for use in typing pools and secretarial offices. Perhaps they had in mind "dedicated", as distinct from "stand-alone"; that's what eventuated in various cases, most notably in the legal/patent department and in the AT&T CEO's office.
>
> Both those systems were targets of opportunity, not foreseen from the start. When Unix was up and running on the PDP-11, Joe got wind of the legal department having installed a commercial word processor. He went to pitch Unix as an alternative and clinched a trial by promising to make `roff` able to number lines by tomorrow in order to fulfill a patent-office requirement that the commercial system did not support.
>
> Modems were installed so legal-department secretaries could try the Research machine. They liked it and Joe's superb customer service. Soon the legal department got a system of their own. Joe went on to create `nroff` and `troff`. Document preparation became a widespread use of Unix, but no stand-alone word-processing system was ever undertaken.

A history relating `groff` to its predecessors `roff`, `nroff`, and `troff` is available in the *roff*(7) man page.

## 1.2 What Is `groff`?

`groff` (GNU `roff`) is a typesetting system that reads plain text input files that include formatting commands to produce output in PostScript, PDF, HTML, DVI, or other formats, or for display to a terminal. Formatting commands can be low-level typesetting primitives, macros from a supplied package, or user-defined macros. All three approaches can be combined.

A reimplementation and extension of the typesetter from AT&T Unix, `groff` is present on most POSIX systems owing to its long association with Unix manuals (including man pages). It and its predecessor are notable for

their production of several best-selling software engineering texts. `groff` is capable of producing typographically sophisticated documents while consuming minimal system resources.

## 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, adjustment, 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
- 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, such as starting paragraphs, or printing headers and footers, 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 Installation

Installation procedures are documented by the files `INSTALL`, `INSTALL.extra`, and `INSTALL.REPO` in the `groff` source distribution.

## 1.8 Conventions Used in This Manual

We apply the term "groff" to the language documented here, the GNU implementation of the overall system, the project that develops that system, and the command of that name. In the first sense, `groff` is an extended dialect of the `roff` language, for which many similar implementations exist.

The `roff` language features several major categories for which many items are predefined. Presentations of these items feature the form in which the

item is most commonly used on the left, and, aligned to the right margin, the name of the category in brackets.

`\n[example]` [Register]

The register '`example`' is one that that `groff` *doesn't* predefine. You can create it yourself, though; see Section 5.8.1 [Setting Registers], page 94.

To make this document useful as a reference and not merely amiable bedtime reading, we tend to present these syntax items in exhaustive detail when they arise. References to topics discussed later in the text are frequent; skip material you don't understand yet.

We use Texinfo's "result" (⇒) and error notations to present output written to the standard output and standard error streams, respectively. Diagnostic messages from the GNU `troff` formatter and other programs are examples of the latter, but the formatter can also be directed to write user-specified messages to the standard error stream. The notation then serves to identify the output stream and does not necessarily mean that an error has occurred.[1]

```
$ echo "Twelve o'clock and" | groff -Tascii | sed '/^$/d'
    ⇒ Twelve o'clock and
$ echo '.tm all is well.' | groff > /dev/null
    error  all is well.
```

Sometimes we use ⇒ somewhat abstractly to represent formatted text that you will need to use a PostScript or PDF viewer program (or a printer) to observe. While arguably an abuse of notation, we think this preferable to requiring the reader to understand the syntax of these page description languages.

We also present diagnostic messages in an abbreviated form, often omitting the name of the program issuing them, the input file name, and line number or other positional information when such data do not serve to illuminate the topic under discussion.

Most examples are of `roff` language input that would be placed in a text file. Occasionally, we start an example with a '`$`' character to indicate a shell prompt, as seen above.

You are encouraged to try the examples yourself, and to alter them to better learn `groff`'s behavior. Our examples frequently need to direct the formatter to set a line length (with '`.ll`') that will fit within the page margins of this manual. We mention this so that you know why it is there before we discuss the `ll` request formally.[2]

---

[1] Unix and related operating systems distinguish standard output and standard error streams *because* of `troff`: `https://minnie.tuhs.org/pipermail/tuhs/2013-December/006113.html`.

[2] See Section 5.15 [Line Layout], page 124.

## 1.9 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 much of the material on the `ms` macro package.

# 2 Invoking groff

This chapter 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 preprocessors and output drivers can be found in the man pages *gpic*(1), *geqn*(1), *gtbl*(1), *ggrn*(1), *grefer*(1), *gchem*(1), *gsoelim*(1), *preconv*(1), *grotty*(1), *grops*(1), *gropdf*(1), *grohtml*(1), *grodvi*(1), *grolj4*(1), *grolbp*(1), and *gxditview*(1).

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... ]
```

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**.

[1] Besides **groff**, **neatroff** is an exception.

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. See its man page *grog*(1); type '`man grog`' at the command line to view it.

`groff`'s command-line options are as follows.

'`-a`' Generate a plain text approximation of the typeset output. The read-only register `.A` is set to 1. See Section 5.8.5 [Built-in Registers], page 100. This option produces a sort of abstract preview of the formatted output.

- Page breaks are marked by a phrase in angle brackets; for example, '`<beginning of page>`'.
- Lines are broken where they would be in the formatted output.
- A horizontal motion of any size is represented as one space. Adjacent horizontal motions are not combined. Intersentence space nodes (those arising from the second argument to the `ss` request) are not represented.
- Vertical motions are not represented.
- Special characters are rendered in angle brackets; for example, the default soft hyphen character appears as '`<hy>`'.

The above description should not be considered a specification; the details of `-a` output are subject to change.

'`-b`' Write a backtrace reporting the state of `gtroff`'s input parser to the standard error stream with each diagnostic message. The line numbers given in the backtrace might not always be correct, because `gtroff`'s idea of line numbers can be confused by requests that append to macros.

'`-c`' Start with color output disabled.

'`-C`' Enable AT&T `troff` compatibility mode; implies `-c`. See Section 5.38 [Implementation Differences], page 224, for the list of incompatibilities between `groff` and AT&T `troff`.

'`-dctext`'
'`-dstring=text`'
Define `roff` string *c* or *string* as *t* or *text*. *c* must be one character; *string* can be of arbitrary length. Such string assignments happen before any macro file is loaded, including the startup file. Due to `getopt_long` limitations, *c* cannot be, and *string* cannot contain, an equals sign, even though that is a valid character in a `roff` identifier.

'-D*enc*' Set fallback input encoding used by `preconv` to *enc*; implies `-k`.

'`-e`' Run `geqn` preprocessor.

'`-E`' Inhibit `gtroff` error messages. This option does *not* suppress messages sent to the standard error stream by documents or macro packages using `tm` or related requests.

'-f*fam*' Use *fam* as the default font family. See Section 5.19.2 [Font Families], page 133.

'-F*dir*' Search in directory *dir* for the selected output device's directory of device and font description files. See the description of `GROFF_FONT_PATH` in Section 2.2 [Environment], page 12, below for the default search locations and ordering.

'`-g`' Run `ggrn` preprocessor.

'`-G`' Run `grap` preprocessor; implies `-p`.

'`-h`' Display a usage message and exit.

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

'-I*dir*' Search the directory *dir* for files named in several contexts; implies `-g` and `-s`.

- `gsoelim` replaces `so` requests with the contents of their file name arguments.
- `gtroff` searches for files named as operands in its command line and as arguments to `psbb`, `so`, and `soquiet` requests.
- Output drivers may search for files; for instance, `grops` looks for files named in '`\X'ps: import ...'`', '`\X'ps: file ...'`', and '`\X'pdf: pdfpic ...'`' device control escape sequences.

This option may be specified more than once; the directories are searched in the order specified. If you want to search the current directory before others, add '`-I .`' at the desired place. The current working directory is otherwise searched last. `-I` works similarly to, and is named for, the "include" option of Unix C compilers.

`-I` options are passed to `gsoelim`, `gtroff`, and output drivers; with the flag letter changed to `-M`, they are also passed to `ggrn`.

'`-j`' Run `gchem` preprocessor. Implies `-p`.

'`-k`' Run `preconv` preprocessor. Refer to its man page for its behavior if neither of `groff`'s `-K` or `-D` options is also specified.

'-K*enc*' Set input encoding used by `preconv` to *enc*; implies `-k`.

'`-l`' Send the output to a spooler for printing. The `print` directive in the device description file specifies the default command to be used; see Section 6.2 [Device and Font Description Files], page 244. See options `-L` and `-X`.

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

'`-m`*name*' Process the file *name*`.tmac` prior to any input files. If not found, `tmac.`*name* is attempted. *name* (in both arrangements) is presumed to be a macro file; see the description of `GROFF_TMAC_PATH` in Section 2.2 [Environment], page 12, below for the default search locations and ordering. This option and its argument are also passed to `geqn`, `grap`, and `ggrn`.

'`-M`*dir*' Search directory *dir* for macro files; see the description of `GROFF_TMAC_PATH` in Section 2.2 [Environment], page 12, below for the default search locations and ordering. This option and its argument are also passed to `geqn`, `grap`, and `ggrn`.

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

'`-N`' Prohibit newlines between `eqn` delimiters: pass `-N` to `geqn`.

'`-o`*list*' Output only pages in *list*, which is a comma-separated list of page ranges; '*n*' means page *n*, '*m*`-`*n*' means every page between *m* and *n*, '`-`*n*' means every page up to *n*, '*n*`-`' means every page from *n* on. `gtroff` stops processing and exits after formatting the last page enumerated in *list*.

'`-p`' Run `gpic` preprocessor.

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

'`-r`*cnumeric-expression*'
'`-r`*register*`=`*expr*'
Set `roff` register *c* or *register* to the value *numeric-expression* (see Section 5.4 [Numeric Expressions], page 79). *c* must be one character; *register* can be of arbitrary length. Such register assignments happen before any macro file is loaded, including the startup file. Due to `getopt_long` limitations, *c* cannot be, and *register* cannot contain, an equals sign, even though that is a valid character in a `roff` identifier.

'`-R`' Run `grefer` preprocessor. No mechanism is provided for passing arguments to `grefer` because most `grefer` options have equivalent language elements that can be specified within the document.

`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`' Run `gsoelim` preprocessor.

'`-S`' Operate in "safer" mode; see `-U` below for its opposite. For security reasons, safer mode is enabled by default.

'`-t`' Run `gtbl` preprocessor.

'`-T`*dev*' Direct `gtroff` to format the input for the output device *dev*. `groff` then calls an output driver to convert `gtroff`'s output to a form appropriate for *dev*. The following output devices are 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-point base font in the document.

`X100` For a 100 dpi X11 previewer.

`X100-12` For a 100 dpi X11 previewer with a 12-point 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 GNU `troff` string `.T` contains the name of the output device; the read-only register `.T` is set to 1 if this option is used (which is always true if `groff` is used to call GNU `troff`). See Section 5.8.5 [Built-in Registers], page 100.

The postprocessor to be used for a device is specified by the `postpro` command in the device description file. (See Section 6.2 [Device and Font Description Files], page 244.) This can be overridden with the `-X` option.

'`-U`' Operate in *unsafe mode*, which enables the `open`, `opena`, `pi`, `pso`, and `sy` requests. These requests are disabled by default because they allow an untrusted input document to write to arbitrary file names and run arbitrary commands. This option also adds the current directory to the macro package search path; see the `-m` option above. `-U` is passed to `gpic` and `gtroff`.

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

'`-V`' Output the pipeline that would be run by `groff` (as a wrapper program) to the standard output stream, but do not execute it. If given more than once, the pipeline is both written to the standard error stream and run.

'`-wcategory`'
Enable warnings in *category*. Categories are listed in Section 5.37.1 [Warnings], page 222.

'`-Wcategory`'
Inhibit warnings in *category*. Categories are listed in Section 5.37.1 [Warnings], page 222.

'`-X`' Use `gxditview` instead of the usual postprocessor to (pre)view a document on an X11 display. Combining this option with `-Tps` uses the font metrics of the PostScript device, whereas the `-TX75` and `-TX100` options use the metrics of X11 fonts.

'`-z`' Suppress formatted output from `gtroff`.

'`-Z`' Disable postprocessing. `gtroff` output will appear on the standard output stream (unless suppressed with `-z`; see Section 6.1 [gtroff Output], page 231, for a description of this format.

## 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 variable is passed to the preconv preprocessor's -e option to select the character encoding of input files. This variable's existence implies the groff option -k. If set but empty, groff calls preconv without an -e option. groff's -K option overrides GROFF_ENCODING. See the *preconv*(7) man page; type 'man preconv' at the command line to view it.

GROFF_FONT_PATH
: A list of directories in which to seek the selected output device's directory of device and font description files. GNU troff will search directories given as arguments to any specified -F options before these, and a built-in list of directories after them. See Section 2.4 [Font Directories], page 14, and the *troff*(1) or *gtroff*(1) man pages.

GROFF_TMAC_PATH
: A list of directories in which to seek macro files. GNU troff will search directories given as arguments to any specified -M options before these, and a built-in list of directories after them. See Section 2.3 [Macro Directories], page 14, and the *troff*(1) or *gtroff*(1) man pages.

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
: Sets the default output device. If empty or not set, a build-time default (often ps) is used. The -T*dev* option overrides GROFF_TYPESETTER.

`SOURCE_DATE_EPOCH`
: A timestamp (expressed as seconds since the Unix epoch) to use as the output creation timestamp in place of the current time. The time is converted to human-readable form using *localtime*(3) when the formatter starts up and stored in registers usable by documents and macro packages (see Section 5.8.5 [Built-in Registers], page 100).

`TZ`
: The time zone to use when converting the current time (or value of `SOURCE_DATE_EPOCH`) to human-readable form; see *tzset*(3).

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 5.19.2 [Font Families], page 133), 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 Format

In groff, the page dimensions for the formatter GNU troff and for output devices are handled separately. See Section 5.17 [Page Layout], page 128, for vertical manipulation of the page size, and See Section 5.15 [Line Layout], page 124, for horizontal changes. The papersize macro package, normally loaded by troffrc at startup, provides an interface for configuring page dimensions by convenient names, like 'letter' or 'a4'; see *groff_tmac*(5). The default used by the formatter depends on its build configuration, but is usually one of the foregoing, as geographically appropriate.

It is up to each macro package to respect the page dimensions configured in this way.

For each output device, the size of the output medium can be set in its DESC file. Most output drivers also recognize a command-line option -p to override the default dimensions and an option -l to use landscape orientation. See Section 6.2.1 [DESC File Format], page 244, for a description of the papersize keyword, which takes an argument of the same form as -p. The output driver's man page, such as *grops*(1), may also be helpful.

groff uses the command-line option -P to pass options to postprocessors; for example, use the following for PostScript output on A4 paper in landscape orientation.

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

## 2.6 Invocation Examples

`roff` systems are best known for formatting man pages. Once a `man` librarian program has located a man page, it may execute a `groff` command much like the following.

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

The librarian will also pipe the output through a pager, which might not interpret the SGR terminal escape sequences `groff` emits for boldface, underlining, or italics; see the *grotty*(1) man page for a discussion.

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

```
gpic foo.me | gtbl | gtroff -me -Tutf8 | grotty
```

Using `groff`, this pipe can be shortened to an equivalent command.

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

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

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

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

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

and

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

are equivalent.

# 3 Tutorial for Macro Users

Most users of the `roff` language employ a macro package to format their documents. Successful macro packages tend to ease the composition process; their users need not have mastered the full formatting language, nor even some of its major features like diversions, traps, and environments. A familiarity with some basic concepts and mechanisms common to macro packages (like "displays") remains helpful; this chapter aims to bring you to this level. If you prefer a meticulous and comprehensive presentation, try Chapter 5 [GNU troff Reference], page 65, instead.

## 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.

GNU `troff` 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 *escape sequences*), which tell GNU `troff` how to format the output. For more detail on this, see Section 5.6 [Formatter Instructions], page 85.

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). See Section 5.6.2 [Invoking Requests], page 86.

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.

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

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

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

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

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 inter-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 5.3 [Measurements], page 77).

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

[2] If you need finer granularity of the vertical space, use the `pvs` request (see Section 5.20.1 [Changing the Type Size], page 151).

'`.ce N`' (where *N* is the number of lines to center), followed by the *N* lines. To center many lines without counting them, type:

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

The '`.ce 0`' request tells `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 5.17 [Page Layout], page 128).

### 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 surveys the "major" macro packages that come with `groff`. One, `ms`, is presented in detail.

Major macro packages are also sometimes described as *full-service* due to the breadth of features they provide and because more than one cannot be used by the same document; 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
```

Many auxiliary, or "minor" macro packages are also available. They may in general be used with any full-service macro package and handle a variety of tasks from character encoding selection, to language localization, to inlining of raster images. See the *groff_tmac*(5) man page for a list. Type '`man groff_tmac`' at the command line to view it.

## 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` 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 a "constant-width" (monospaced) typeface (Courier roman).

`.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 a monospaced typeface (Courier roman). 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 monospaced typeface (Courier roman), 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 monospaced typeface (Courier roman), 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") package is suitable for the preparation of letters, memoranda, reports, and books. These groff macros feature cover page and table of contents generation, automatically numbered headings, several paragraph styles, a variety of text styling options, footnotes, and multi-column page layouts. ms supports the tbl, eqn, pic, and refer preprocessors for inclusion of tables, mathematical equations, diagrams, and standardized bibliographic citations. This implementation is mostly compatible with the documented interface and behavior of AT&T Unix Version 7 ms. Many extensions from 4.2BSD (Berkeley) and Tenth Edition Research Unix have been recreated.

### 4.6.1 Introduction

The `ms` macros are the oldest surviving package for `roff` systems.[1] While the `man` package was designed for brief reference documents, the `ms` macros are also suitable for longer works intended for printing and possible publication.

#### 4.6.1.1 Basic information

`ms` documents are plain text files; prepare them with your preferred text editor. If you're in a hurry to start, know that `ms` needs one of its macros called at the beginning of a document so that it can initialize. A *macro* is a formatting instruction to `ms`. Put a macro call on a line by itself. Use '`.PP`' if you want your paragraph's first line to be indented, or '`.LP`' if you don't.

After that, start typing normally. It is a good practice to start each sentence on a new line, or to put two spaces after sentence-ending puncutation, so that the formatter knows where the sentence boundaries are. You can separate paragraphs with further paragraphing 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 part of the manual.

Format the document with the `groff` command. `nroff` can be useful for previewing.

```
$ editor radical.ms
$ nroff -ww -z -ms radical.ms # check for errors
$ nroff -ms radical.ms | less -R
$ groff -T ps -ms radical.ms > radical.ps
$ see radical.ps
```

Our `radical.ms` document might look like this.

```
.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.
```

`ms` exposes many aspects of document layout to user control via `groff`'s *registers* and *strings*, which store numbers and text, respectively. Measurements in `groff` are expressed with a suffix called a *scaling unit*.

[1] While manual *pages* are older, early ones used macros supplanted by the `man` package of Seventh Edition Unix (1979). `ms` shipped with Sixth Edition (1975) and was documented by Mike Lesk in a Bell Labs internal memorandum.

| | |
|---|---|
| `i` | inches |
| `c` | centimeters |
| `p` | points (1/72 inch) |
| `P` | picas (1/6 inch) |
| `v` | vees; current vertical spacing |
| `m` | ems; width of an "M" in the current font |
| `n` | ens; one-half em |

Set registers with the `nr` request and strings with the `ds` request. *Requests* are like macro calls; they go on lines by themselves and start with the *control character*, a dot (`.`). The difference is that they directly instruct the formatter program, rather than the macro package. We'll discuss a few as applicable. It is wise to specify a scaling unit when setting any register that represents a length, size, or distance.

```
.nr PS 10.5p \" Use 10.5-point type.
.ds FAM P    \" Use Palatino font family.
```

In the foregoing, we see that `\"` begins a comment. This is an example of an *escape sequence*, the other kind of formatting instruction. Escape sequences can appear anywhere. They begin with the escape character (`\`) and are followed by at least one more character. `ms` documents tend to use only a few of `groff`'s many requests and escape sequences; see Appendix B [Request Index], page 261, and Appendix C [Escape Sequence Index], page 265, or the *groff*(7) man page for complete lists.

| | |
|---|---|
| `\"` | Begin comment; ignore remainder of line. |
| `\n[`*reg*`]` | Interpolate value of register *reg*. |
| `\*[`*str*`]` | Interpolate contents of string *str*. |
| `\*`*s* | abbreviation of `\*[s]`; the name *s* must be only one character |
| `\[`*char*`]` | Interpolate glyph of special character named *char*. |
| `\&` | dummy character |
| `\~` | Insert an unbreakble space that is adjustable like a normal space. |
| `\|` | Move horizontally by one-sixth em ("thin space"). |

Prefix any words that start with a dot '`.`' or neutral apostrophe '`'`' with `\&` if they are at the beginning of an input line (or might become that way in editing) to prevent them from being interpreted as macro calls or requests. Suffix '`.`', '`?`', and '`!`' with `\&` when needed to cancel end-of-sentence detection.

```
My exposure was \&.5 to \&.6 Sv of neutrons, said Dr.\&
Wallace after the criticality incident.
```

### 4.6.2 Document Structure

The `ms` macro package expects a certain amount of structure: a well-formed document contains at least one paragraphing or heading macro call. Longer documents have a structure as follows.

**Document type**
: Calling the `RP` macro at the beginning of your document puts the document description (see below) on a cover page. Otherwise, `ms` places the information (if any) on the first page, followed immediately by the body text. Some document types found in other `ms` implementations are specific to AT&T or Berkeley, and are not supported by `groff ms`.

**Format and layout**
: By setting registers and strings, you can configure your document's typeface, margins, spacing, headers and footers, and footnote arrangement. See Section 4.6.3 [ms Document Control Settings], page 30.

**Document description**
: A document description consists of any of: a title, one or more authors' names and affiliated institutions, an abstract, and a date or other identifier. See Section 4.6.4 [ms Document Description Macros], page 35.

**Body text**
: The main matter of your document follows its description (if any). `ms` supports highly structured text consisting of paragraphs interspersed with multi-level headings (chapters, sections, subsections, and so forth) and augmented by lists, footnotes, tables, diagrams, and similar material. See Section 4.6.5 [ms Body Text], page 37.

**Tables of contents**
: Macros enable the collection of entries for a table of contents (or index) as the material they discuss appears in the document. You then call a macro to emit the table of contents at the end of your document. The table of contents must necessarily follow the rest of the text since GNU `troff` is a single-pass formatter; it thus cannot determine the page number of a division of the text until it has been set and output. Since `ms` was designed for the production of hard copy, the traditional procedure was to manually relocate the pages containing the table of contents between the cover page and the body text. Today,

page resequencing can be done in the digital domain with tools like *pdfjam*(1). An index works similarly, but because it typically needs to be sorted after collection, its preparation requires separate processing.

### 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.

A list of document control registers and strings follows. For any parameter whose default is unsatisfactory, define its register or string before calling any `ms` macro other than `RP`.

#### Margin settings

`\n[PO]` [Register]
Defines the page offset (i.e., the left margin).

Effective: next page.

Default: Varies by output device and paper format; 1 i is used for typesetters using U.S. letter paper, and zero for terminals. See Section 2.5 [Paper Format], page 15.

`\n[LL]` [Register]
Defines the line length (i.e., the width of the body text).

Effective: next paragraph.

Default: Varies by output device and paper format; 6.5 i is used for typesetters using U.S. letter paper (see Section 2.5 [Paper Format], page 15) and 65 n on terminals.

`\n[LT]` [Register]
Defines the title line length (i.e., the header and footer width). This is usually the same as `LL`, but need not be.

Effective: next paragraph.

Default: Varies by output device and paper format; 6.5 i is used for typesetters using U.S. letter paper (see Section 2.5 [Paper Format], page 15) and 65 n on terminals.

`\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.

## Titles (headers, footers)

`\*[LH]` [String]
Defines the text displayed in the left header position.
Effective: next header.
Default: empty.

`\*[CH]` [String]
Defines the text displayed in the center header position.
Effective: next header.
Default: '`-\n[%]-`'.

`\*[RH]` [String]
Defines the text displayed in the right header position.
Effective: next header.
Default: empty.

`\*[LF]` [String]
Defines the text displayed in the left footer position.
Effective: next footer.
Default: empty.

`\*[CF]` [String]
Defines the text displayed in the center footer position.
Effective: next footer.
Default: empty.

`\*[RF]` [String]
Defines the text displayed in the right footer position.
Effective: next footer.
Default: empty.

## Text settings

`\n[PS]` [Register]
Defines the type size of the body text.
Effective: next paragraph.
Default: 10 p.

`\n[VS]` [Register]
Defines the vertical spacing (type size plus leading).
Effective: next paragraph.
Default: 12 p.

`\n[HY]` [Register]
Defines the automatic hyphenation mode used with the `hy` request. Setting `HY` to 0 is equivalent to using the `nh` request. This is a Tenth Edition Research Unix extension.
Effective: next paragraph.
Default: 6.

`\*[FAM]` [String]
Defines the font family used to typeset the document. This is a GNU extension.
Effective: next paragraph.
Default: defined by the output device; often '`T`' (see Section 4.6.5 [ms Body Text], page 37)

## Paragraph settings

`\n[PI]` [Register]
Defines the indentation amount used by the `PP`, `IP` (unless overridden by an optional argument), `XP`, and `RS` macros.
Effective: next paragraph.
Default: 5 n.

`\n[PD]` [Register]
Defines the space between paragraphs.
Effective: next paragraph.
Default: 0.3 v (1 v on low-resolution devices).

`\n[QI]` [Register]
Defines the indentation amount used on both sides of a paragraph set with the `QP` or between the `QS` and `QE` macros.
Effective: next paragraph.
Default: 5 n.

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

Default: 1.

### Heading settings

`\n[PSINCR]` [Register]
Defines an increment in type size to be applied to a heading at a lesser depth than that specified in `GROWPS`. The value of `PSINCR` should be specified in points with the p scaling unit and may include a fractional component; for example, '`.nr PSINCR 1.5p`' sets a type size increment of 1.5 p. This is a GNU extension.

Effective: next heading.

Default: 1 p.

`\n[GROWPS]` [Register]
Defines the heading depth above which the type size increment set by `PSINCR` becomes effective. For each heading depth less than the value of `GROWPS`, the type size is increased by `PSINCR`. Setting `GROWPS` to any value less than 2 disables the incremental heading size feature. This is a GNU extension.

Effective: next heading.

Default: 0.

`\n[HORPHANS]` [Register]
Defines the minimum number of lines of an immediately succeeding paragraph that should be kept together with any heading introduced by the `NH` or `SH` macros. If a 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. This is a GNU extension.

Effective: next paragraph.

Default: 1.

`\*[SN-STYLE]` [String]
Defines the style used to print numbered headings. See Section 4.6.5.4 [Headings in ms], page 40. This is a GNU extension.

Effective: next heading.

Default: alias of `SN-DOT`

### Footnote settings

`\n[FI]` [Register]
Defines the footnote indentation. This is a Berkeley extension.

Effective: next footnote.

Default: 2 n.

`\n[FF]` [Register]
Defines the format of automatically numbered footnotes, and those for which the `FS` request is given a marker argument, at the bottom of a column or page. This is a Berkeley extension.

`0` Set an automatic number[2] as a superscript (on typesetter devices) or surrounded by square brackets (on terminals). The footnote paragraph is indented as with `PP` if there is an `FS` argument or an automatic number, and as with `LP` otherwise. This is the default.

`1` As `0`, but set the marker as regular text and follow an automatic number with a period.

`2` As `1`, but without indentation (like `LP`).

`3` As `1`, but set the footnote paragraph with the marker hanging (like `IP`).

Effective: next footnote.

Default: 0.

`\n[FPS]` [Register]
Defines the footnote type size.

Effective: next footnote.

Default: `\n[PS] - 2p`.

`\n[FVS]` [Register]
Defines the footnote vertical spacing.

Effective: next footnote.

Default: `\n[FPS] + 2p`.

`\n[FPD]` [Register]
Defines the footnote paragraph spacing. This is a GNU extension.

Effective: next footnote.

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

`\*[FR]` [String]
Defines the ratio of the footnote line length to the current line length. This is a GNU extension.

Effective: next footnote in single-column arrangements, next page otherwise.

Default: `11/12`.

[2] defined in Section 4.6.5.10 [ms Footnotes], page 51

### Display settings

`\n[DD]` [Register]
Sets the display distance—the vertical spacing before and after a display, a `tbl` table, an `eqn` equation, or a `pic` image. This is a Berkeley extension.

Effective: next display boundary.

Default: 0.5 v (1 v on low-resolution devices).

`\n[DI]` [Register]
Sets the default amount by which to indent a display started with `DS` and `ID` without arguments, to '`.DS I`' without an indentation argument, and to equations set with '`.EQ I`'. This is a GNU extension.

Effective: next indented display.

Default: 0.5 i.

### Other settings

`\n[MINGW]` [Register]
Defines the default minimum width between columns in a multi-column document. This is a GNU extension.

Effective: next page.

Default: 2 n.

`\n[TC-MARGIN]` [Register]
Defines the width of the field in which page numbers are set in a table of contents entry; the right margin thus moves inboard by this amount. This is a GNU extension.

Effective: next `PX` call.

Default: `\w'000'`

## 4.6.4 Document Description Macros

Only the simplest document lacks a title.[3] As its level of sophistication (or complexity) increases, it tends to acquire a date of revision, explicitly identified authors, sponsoring institutions for authors, and, at the rarefied heights, an abstract of its content. Define these data by calling the macros below in the order shown; `DA` or `ND` can be called to set the document date (or other identifier) at any time before (a) the abstract, if present, or (b) its information is required in a header or footer. Use of these macros is optional, except that `TL` is mandatory if any of `RP`, `AU`, `AI`, or `AB` is called, and `AE` is mandatory if `AB` is called.

`.RP [no-repeat-info] [no-renumber]` [Macro]
Use the "report" (AT&T: "released paper") format for your document, creating a separate cover page. The default arrangement is to place most

[3] Distinguish a document title from "titles", which are what `roff` systems call headers and footers collectively.

of the document description (title, author names and institutions, and abstract, but not the date) at the top of the first page. If the optional `no-repeat-info` argument is given, `ms` produces a cover page but does not repeat any of its information subsequently (but see the `DA` macro below regarding the date). Normally, `RP` sets the page number following the cover page to 1. Specifying the optional `no-renumber` argument suppresses this alteration. Optional arguments can occur in any order. `no` is recognized as a synonym of `no-repeat-info` for `AT&T` compatibility.

`.TL` [Macro]
Specify the document title. `ms` collects text on input lines following this call into the title until reaching `AU`, `AB`, or a heading or paragraphing macro call.

`.AU` [Macro]
Specify an author's name. `ms` collects text on input lines following this call into the author's name until reaching `AI`, `AB`, another `AU`, or a heading or paragraphing macro call. Call it repeatedly to specify multiple authors.

`.AI` [Macro]
Specify the preceding author's institution. An `AU` call is usefully followed by at most one `AI` call; if there are more, the last `AI` call controls. `ms` collects text on input lines following this call into the author's institution until reaching `AU`, `AB`, or a heading or paragraphing macro call.

`.DA` [*x* ...] [Macro]
Typeset the current date, or any arguments *x*, in the center footer, and, if `RP` is also called, left-aligned at the end of the description information on the cover page.

`.ND` [*x* ...] [Macro]
Typeset the current date, or any arguments *x*, if `RP` is also called, left-aligned at the end of the document description on the cover page. This is `groff ms`'s default.

`.AB` [`no`] [Macro]
Begin the abstract. `ms` collects text on input lines following this call into the abstract until reaching an `AE` call. By default, `ms` places the word "ABSTRACT" centered and in italics above the text of the abstract. The optional argument `no` suppresses this heading.

`.AE` [Macro]
End the abstract.

An example document description, using a cover page, follows.

```
.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 more in line with user demand.
.AE

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

### 4.6.5 Body Text

A variety of macros, registers, and strings can be used to structure and style the body of your document. They organize your text into paragraphs, headings, footnotes, and inclusions of material such as tables and figures.

#### 4.6.5.1 Text settings

The `FAM` string, a GNU extension, sets the font family for body text; the default is '`T`'. The `PS` and `VS` registers set the type size and vertical spacing (distance between text baselines), respectively. The font family and type size are ignored on terminal devices. Setting these parameters before the first call of a heading, paragraphing, or (non-date) document description macro also applies them to headers, footers, and (for `FAM`) footnotes.

Which font families are available depends on the output device; as a convention, `T` selects a serif family ("Times"), `H` a sans-serif family ("Helvetica"), and `C` a monospaced family ("Courier"). The man page for the

output driver documents its font repertoire. Consult the *groff*(1) man page for lists of available output devices and their drivers.

The hyphenation mode (as used by the `hy` request) is set from the `HY` register. Setting `HY` to '`0`' is equivalent to using the `nh` request. This is a Tenth Edition Research Unix extension.

### 4.6.5.2 Typographical symbols

`ms` provides a few strings to obtain typographical symbols not easily entered with the keyboard. These and many others are available as special character escape sequences—see the *groff_char*(7) man page.

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

`\*[Q]` [String]
`\*[U]` [String]
Interpolate typographer's quotation marks where available, and neutral double quotes otherwise. `\*Q` is the left quote and `\*U` the right.

### 4.6.5.3 Paragraphs

Paragraphing macros *break*, or terminate, any pending output line so that a new paragraph can begin. Several paragraph types are available, differing in how indentation applies to them: to left, right, or both margins; to the first output line of the paragraph, all output lines, or all but the first. All paragraphing macro calls cause the insertion of vertical space in the amount stored in the `PD` register, except at page or column breaks. Alternatively, a blank input line breaks the output line and vertically spaces by one vee.

`.LP` [Macro]
Set a paragraph without any (additional) indentation.

`.PP` [Macro]
Set a paragraph with a first-line left indentation in the amount stored in the `PI` register.

`.IP` [*marker* [*width*]] [Macro]
Set a paragraph with a left indentation. The optional *marker* is not indented and is empty by default. It has several applications; see Section 4.6.5.6 [Lists in ms], page 44. *width* overrides the indentation amount stored in the `PI` register; its default unit is '`n`'. Once specified, *width* applies to further `IP` calls until specified again or a heading or different paragraphing macro is called.

`.QP` [Macro]
Set a paragraph indented from both left and right margins by the amount stored in the `QI` register.

`.QS` [Macro]
`.QE` [Macro]
Begin (`QS`) and end (`QE`) a region where each paragraph is indented from both margins by the amount stored in the `QI` register. The text between `QS` and `QE` can be structured further by use of other paragraphing macros.

`.XP` [Macro]
Set an "exdented" paragraph—one with a left indentation in the amount stored in the `PI` register on every line *except* the first (also known as a hanging indent). This is a Berkeley extension.

The following example illustrates the use of paragraphing macros.

```
.NH 2
Cases used in the 2001 study
.LP
Two software releases were considered for this report.
.PP
The first is commercial software;
the second is free.
.IP \[bu]
Microsoft Word for Windows,
starting with version 1.0 through the current version
(Word 2000).
.IP \[bu]
GNU 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.
.SH
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.
```

#### 4.6.5.4 Headings

Use headings to create a sequential or hierarchical structure for your document. The `ms` macros print headings in **bold** using the same font family and, by default, type size as the body text. Headings are available with and without automatic numbering. Text on input lines following the macro call becomes the heading's title.

`.NH` [*depth*] [Macro]
`.NH S` *heading-depth-index . . .* [Macro]
Set an automatically numbered heading.

`ms` produces a numbered heading the form *a.b.c. . .*, to any depth desired, with the numbering of each depth increasing automatically and being reset to zero when a more significant level is increased. "1" is the most significant or coarsest division of the document. Only nonzero values are output. If *depth* is omitted, it is taken to be '`1`'.

If you specify *depth* such that an ascending gap occurs relative to the previous `NH` call—that is, you "skip a depth", as by '`.NH 1`' and then '`.NH 3`'—`groff ms` emits a warning on the standard error stream.

Alternatively, you can give `NH` a first argument of `S`, followed by integers to number the heading depths explicitly. Further automatic numbering, if used, resumes using the specified indices as their predecessors. This feature is a Berkeley extension.

An example may be illustrative.

```
.NH 1
Animalia
.NH 2
Arthropoda
.NH 3
Crustacea
.NH 2
Chordata
.NH S 6 6 6
Daimonia
.NH 1
Plantae
```

The above results in numbering as follows; the vertical space that normally precedes each heading is omitted.

```
1.  Animalia
1.1.  Arthropoda
1.1.1.  Crustacea
1.2.  Chordata
6.6.6.  Daimonia
7.  Plantae
```

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

After `NH` is called, the assigned number is made available in the strings `SN-DOT` (as it appears in a printed heading with default formatting, followed by a terminating period) and `SN-NO-DOT` (with the terminating period omitted). These are GNU extensions.

You can control the style used to print numbered headings by defining an appropriate alias for the string `SN-STYLE`. By default, `SN-STYLE` is aliased to `SN-DOT`. If you prefer to omit the terminating period from numbers appearing in numbered headings, you may define the alias as follows.

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

Any such change in numbering style becomes effective from the next use of `NH` following redefinition of the alias for `SN-STYLE`. The formatted number of the current heading is available in the `SN` string (a feature first documented by Berkeley), which facilitates its inclusion in, for example, table captions, equation labels, and `XS`/`XA`/`XE` table of contents entries.

`.SH` [*depth*] [Macro]

Set an unnumbered heading.

The optional *depth* argument is a GNU extension indicating the heading depth corresponding to the *depth* argument of `NH`. It matches the type size at which the heading is set to that of a numbered heading at the same depth when the `GROWPS` and `PSINCR` heading size adjustment mechanism is in effect.

If the `GROWPS` register is set to a value greater than the *level* argument to `NH` or `SH`, the type size of a heading produced by these macros increases by `PSINCR` units over the size specified by `PS` multiplied by the difference of `GROWPS` and *level*. The value stored in `PSINCR` is interpreted in `groff` basic units; the `p` scaling unit should be employed when assigning a value specified in points. For example, the sequence

```
.nr PS 10
.nr GROWPS 3
.nr PSINCR 1.5p
.NH 1
Carnivora
.NH 2
Felinae
.NH 3
Felis catus
.SH 2
Machairodontinae
```

will cause "1. Carnivora" to be printed in 13-point text, followed by "1.1. Felinae" in 11.5-point text, while "1.1.1. Felis catus" and all more deeply nested heading levels will remain in the 10-point text specified by the `PS` register. "Machairodontinae" is printed at 11.5 points, since it corresponds to heading level 2.

The `HORPHANS` register operates in conjunction with the `NH` and `SH` macros to inhibit the printing of isolated headings at the bottom of a page; it specifies the minimum number of lines of an immediately subsequent paragraph that must be kept on the same page as the heading. If insufficient space remains on the current page to accommodate the heading and this number of lines of paragraph text, a page break is forced before the heading is printed. Any display macro call or `tbl`, `pic`, or `eqn` region between the heading and the subsequent paragraph suppresses this grouping. See Section 4.6.5.8 [ms keeps and displays], page 48, and Section 4.6.5.9 [ms Insertions], page 50.

### 4.6.5.5 Typeface and decoration

The `ms` macros provide a variety of ways to style text. Attend closely to the ordering of arguments labeled *pre* and *post*, which is not intuitive. Support for *pre* arguments is a GNU extension.[4]

`.B` [*text* [*post* [*pre*]]] [Macro]
Style *text* in **bold**, followed by *post* in the previous font style without intervening space, and preceded by *pre* similarly. Without arguments, `ms` styles subsequent text in bold until the next paragraphing, heading, or no-argument typeface macro call.

[4] This idiosyncrasy arose through feature accretion; for example, the `B` macro in Version 6 Unix `ms` (1975) accepted only one argument, the text to be set in boldface. By Version 7 (1979) it recognized a second argument; in 1990, `groff ms` added a "pre" argument, placing it third to avoid breaking support for older documents.

`.R` [*text* [*post* [*pre*]]] [Macro]
As `B`, but use the roman style (upright text of normal weight) instead of bold. Argument recognition is a GNU extension.

`.I` [*text* [*post* [*pre*]]] [Macro]
As `B`, but use an *italic* or oblique style instead of bold.

`.BI` [*text* [*post* [*pre*]]] [Macro]
As `B`, but use a bold italic or bold oblique style instead of upright bold. This is a Tenth Edition Research Unix extension.

`.CW` [*text* [*post* [*pre*]]] [Macro]
As `B`, but use a `constant-width` (monospaced) roman typeface instead of bold. This is a Tenth Edition Research Unix extension.

`.BX` [*text*] [Macro]
Typeset *text* and draw a box around it. On terminal devices, reverse video is used instead. If you want *text* to contain space, use unbreakable space or horizontal motion escape sequences (`\~`, `\SP`, `\^`, `\|`, `\0` or `\h`).

`.UL` [*text* [*post*]] [Macro]
Typeset *text* with an underline. *post*, if present, is set after *text* with no intervening space.

`.LG` [Macro]
Set subsequent text in larger type (two points larger than the current size) until the next type size, paragraphing, or heading macro call. You can specify this macro multiple times to enlarge the type size as needed.

`.SM` [Macro]
Set subsequent text in smaller type (two points smaller than the current size) until the next type size, paragraphing, or heading macro call. You can specify this macro multiple times to reduce the type size as needed.

`.NL` [Macro]
Set subsequent text at the normal type size (the amount in the `PS` register).

*pre* and *post* arguments are typically used to simplify the attachment of punctuation to styled words. When *pre* is used, a hyphenation control escape sequence `\%` that would ordinarily start *text* must start *pre* instead to have the desired effect.

```
The CS course's students found one C language keyword
.CW static ) \%(
most troublesome.
```

The foregoing example produces output as follows.

```
The CS course's students found one C language keyword (static)
most troublesome.
```

You can use the output line continuation escape sequence `\c` to achieve the same result (see Section 5.16 [Line Continuation], page 127). It is also portable to older `ms` implementations.

```
The CS course's students found one C language keyword
\%(\c
.CW \%static )
most troublesome.
```

`groff ms` also offers strings to begin and end super- and subscripting. These are GNU extensions.

`\*[{]` [String]
`\*[}]` [String]
Begin and end superscripting, respectively.

`\*[<]` [String]
`\*[>]` [String]
Begin and end subscripting, respectively.

Rather than calling the `CW` macro, in `groff ms` you might prefer to change the font family to Courier 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`'. Because changes to `FAM` take effect only at the next paragraph, `CW` remains useful to "inline" a change to the font family, similarly to the practice of this document in noting syntactical elements of `ms` and `groff`.

#### 4.6.5.6 Lists

The *marker* argument to the `IP` macro can be employed to present a variety of lists; for instance, you can use a bullet glyph (`\[bu]`) for unordered lists, a number (or auto-incrementing register) for numbered lists, or a word or phrase for glossary-style or definition lists. If you set the paragraph indentation register `PI` before calling `IP`, you can later reorder the items in the list without having to ensure that a *width* argument remains affixed to the first call.

The following is an example of a bulleted list.

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

```
A bulleted list:

• lawyers

• guns

• money
```

The following is an example of a numbered list.

```
.nr step 0 1
.nr PI 3n
A numbered list:
.IP \n+[step]
lawyers
.IP \n+[step]
guns
.IP \n+[step]
money
```

```
A numbered list:

1. lawyers

2. guns

3. money
```

Here we have employed the `nr` request to create a register of our own, '`step`'. We initialized it to zero and assigned it an auto-increment of 1. Each time we use the escape sequence '`\n+[PI]`' (note the plus sign), the formatter applies the increment just before interpolating the register's value. Preparing the `PI` register as well enables us to rearrange the list without the tedium of updating macro calls.

The next example illustrates 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!
```

```
A glossary-style list:

lawyers
      Two or more attorneys.

guns  Firearms, preferably large-caliber.

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

In the previous example, observe how the `IP` macro places the definition on the same line as the term if it has enough space. If this is not what you want, there are a few workarounds we will illustrate by modifying the example. First, you can use a `br` request to force a break after printing the term or label.

```
.IP guns
.br
Firearms,
```

Second, you could apply the `\p` escape sequence to force a break. The space following the escape sequence is important; if you omit it, `groff` prints the first word of the paragraph text on the same line as the term or label (if it fits) *then* breaks the line.

```
.IP guns
\p Firearms,
```

Finally, you may append a horizontal motion to the marker with the `\h` escape sequence; using the same amount as the indentation will ensure that the marker is too wide for `groff` to treat it as "fitting" on the same line as the paragraph text.

```
.IP guns\h'0.4i'
Firearms,
```

In each case, the result is the same.

```
A glossary-style list:

lawyers
      Two or more attorneys.

guns
      Firearms, preferably large-caliber.

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

#### 4.6.5.7 Indented regions

You may need to indent a region of text while otherwise formatting it normally. Indented regions can be nested; you can change `\n[PI]` before each call to vary the amount of inset.

`.RS` [Macro]
Begin a region where headings, paragraphs, and displays are indented (further) by the amount stored in the `PI` register.

`.RE` [Macro]
End the (next) most recent indented region.

This feature enables you to easily line up text under hanging and indented paragraphs. For example, you may wish to structure lists hierarchically.

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

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

#### 4.6.5.8 Keeps, boxed keeps, and displays

On occasion, you may want to *keep* several lines of text, or a region of a document, together on a single page, preventing an automatic page break within certain boundaries. This can cause a page break to occur earlier than it normally would. For example, you may want to keep two paragraphs together, or a paragraph that refers to a table, list, or figure adjacent to the item it discusses. `ms` provides the `KS` and `KE` macros for this purpose.

You can alternatively specify a *floating keep*: if a keep cannot fit on the current page, `ms` holds its contents and allows material following the keep (in the source document) to fill the remainder of the current page. When the page breaks, whether by reaching the end or `bp` request, `ms` puts the floating keep at the beginning of the next page. This is useful for placing large graphics or tables that do not need to appear exactly where they occur in the source document.

`.KS` [Macro]
`.KF` [Macro]
`.KE` [Macro]
`KS` begins a keep, `KF` a floating keep, and `KE` ends a keep of either kind.

As an alternative to the keep mechanism, the `ne` request forces a page break if there is not at least the amount of vertical space specified in its argument remaining on the page (see Section 5.18 [Page Control], page 129). One application of `ne` is to reserve space on the page for a figure or illustration to be included later.

A *boxed keep* has a frame drawn around it.

`.B1` [Macro]
`.B2` [Macro]
`B1` begins a keep with a box drawn around it. `B2` ends a boxed keep.

Boxed keep macros cause breaks; if you need to box a word or phrase within a line, see the `BX` macro in Section 4.6.5.5 [Typeface and decoration], page 42. Box lines are drawn as close as possible to the text they enclose so that they are usable within paragraphs. If you wish to box one or more paragraphs, you may improve the appearance by calling `B1` after the first

paragraphing macro, and by adding a small amount of vertical space before calling B2.

```
.LP
.B1
.I Warning:
Happy Fun Ball may suddenly accelerate to dangerous
speeds.
.sp \n[PD]/2 \" space by half the inter-paragraph distance
.B2
```

If you want a boxed keep to float, you will need to enclose the B1 and B2 calls within a pair of KF and KE calls.

*Displays* turn off filling; lines of verse or program code are shown with their lines broken as in the source document without requiring br requests between lines. Displays can be kept on a single page or allowed to break across pages. The DS macro begins a kept display of the layout specified in its first argument; non-kept displays are begun with dedicated macros corresponding to their layout.

.DS L [Macro]
.LD [Macro]
Begin (DS: kept) left-aligned display.

.DS [I [*indent*]] [Macro]
.ID [*indent*] [Macro]
Begin (DS: kept) display indented by *indent* if specified, and by the amount of the DI register otherwise.

.DS B [Macro]
.BD [Macro]
Begin a (DS: kept) a block display: the entire display is left-aligned, but indented such that the longest line in the display is centered on the page.

.DS C [Macro]
.CD [Macro]
Begin a (DS: kept) centered display: each line in the display is centered.

.DS R [Macro]
.RD [Macro]
Begin a (DS: kept) right-aligned display. This is a GNU extension.

.DE [Macro]
End any display.

The distance stored in the DD register is inserted before and after each pair of display macros, replacing any adjacent inter-paragraph distance; this is a

Berkeley extension. The `DI` register is a GNU extension; its value is an indentation applied to displays created with '`.DS`' and '`.ID`' without arguments, to '`.DS I`' without an indentation argument, and to indented equations set with '`.EQ`'. Changes to either register take effect at the next display boundary.

#### 4.6.5.9 Tables, figures, equations, and references

The `ms` package is often used with the `tbl`, `pic`, `eqn`, and `refer` preprocessors. Mark text meant for preprocessors by enclosing it in pairs of tokens as follows, with nothing between the dot and the macro name. The preprocessors match these tokens only at the start of an input line.

`.TS` [`H`] [Macro]
`.TE` [Macro]
Demarcate a table to be processed by the `tbl` preprocessor. The optional argument `H` to `TS` instructs `ms` to repeat table rows (often column headings) at the top of each new page the table spans, if applicable; calling the `TH` macro marks the end of such rows. The GNU *tbl*(1) man page provides a comprehensive reference to the preprocessor and offers examples of its use.

`.PS` [Macro]
`.PE` [Macro]
`.PF` [Macro]
`PS` begins a picture to be processed by the `gpic` preprocessor; either of `PE` or `PF` ends it, the latter with "flyback" to the vertical position at its top. You can create `pic` input manually or with a program such as `xfig`.

`.EQ` [*align* [*label*]] [Macro]
`.EN` [Macro]
Demarcate an equation to be processed by the `eqn` preprocessor. The equation is centered by default; *align* can be '`C`', '`L`', or '`I`' to (explicitly) center, left-align, or indent it by the amount stored in the `DI` register, respectively. If specified, *label* is set right-aligned.

`.[` [Macro]
`.]` [Macro]
Demarcate a bibliographic citation to be processed by the `refer` preprocessor. The GNU *refer*(1) man page provides a comprehensive reference to the preprocessor and the format of its bibliographic database. Type '`man refer`' at the command line to view it.

When `refer` emits collected references (as might be done on a "Works Cited" page), it interpolates the `REFERENCES` string as an unnumbered heading (`SH`).

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

```
.TS H
allbox;
Cb | Cb .
Part→Description
_
.TH
.T&
GH-1978→Fribulating gonkulator
. . . the rest of the table follows. . .
.TE
```

Attempting to place a multi-page table inside a keep can lead to unpleasant results, particularly if the `tbl` `allbox` option is used.

Mathematics can be typeset using the language of the `eqn` preprocessor.

```
.EQ C (\*[SN-NO-DOT]a)
p ~ = ~ q sqrt { ( 1 + ~ ( x / q sup 2 ) }
.EN
```

This input formats a labelled equation. We used the `SN-NO-DOT` string to base the equation label on the current heading number, giving us more flexibility to reorganize the document.

Use `groff` options to run preprocessors on the input: `-e` for `geqn`, `-p` for `gpic`, `-R` for `grefer`, and `-t` for `gtbl`.

#### 4.6.5.10 Footnotes

A footnote is typically anchored to a place in the text with a *marker*, which is a small integer, a symbol such as a dagger, or arbitrary user-specified text.

`\*[*]` [String]
: Place an *automatic number*, an automatically generated numeric footnote marker, in the text. Each time this string is interpolated, the number it produces increments by one. Automatic numbers start at 1. This is a Berkeley extension.

Enclose the footnote text in `FS` and `FE` macro calls to set it at the nearest available "foot", or bottom, of a text column or page.

`.FS` [*marker*] [Macro]
`.FE` [Macro]
: Begin (`FS`) and end (`FE`) a footnote. `FS` calls `FS-MARK` with any supplied *marker* argument, which is then also placed at the beginning of the footnote text. If *marker* is omitted, the next pending automatic footnote

number enqueued by interpolation of the `*` string is used, and if none exists, nothing is prefixed.

You may not desire automatically numbered footnotes in spite of their convenience. You can indicate a footnote with a symbol or other text by specifying its marker at the appropriate place (for example, by using `\[dg]` for the dagger glyph) *and* as an argument to the `FS` macro. Such manual marks should be repeated as arguments to `FS` or as part of the footnote text to disambiguate their correspondence. You may wish to use `\*{` and `\*}` to superscript the marker at the anchor point, in the footnote text, or both.

`groff ms` provides a hook macro, `FS-MARK`, for user-determined operations to be performed when the `FS` macro is called. It is passed the same arguments as `FS` itself. An application of `FS-MARK` is anchor placement for a hyperlink reference, so that a footnote can link back to its referential context.[5] By default, this macro has an empty definition. `FS-MARK` is a GNU extension.

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

Footnote text is formatted as paragraphs are, using analogous parameters. The registers `FI`, `FPD`, `FPS`, and `FVS` correspond to `PI`, `PD`, `PS`, and `CS`, respectively; `FPD`, `FPS`, and `FVS` are GNU extensions.

The `FF` register controls the formatting of automatically numbered footnote paragraphs and those for which `FS` is given a marker argument. See Section 4.6.3 [ms Document Control Settings], page 30.

The default footnote line length is 11/12ths of the normal line length for compatibility with the expectations of historical `ms` documents; you may wish to set the `FR` string to '`1`' to align with contemporary typesetting practices. In the past,[6] an `FL` register was used for the line length in footnotes; however, setting this register at document initialization time had no effect on the footnote line length in multi-column arrangements.[7]

`FR` should be used in preference to the old `FL` register in contemporary documents. The footnote line length is effectively computed as '*`column-width`* `*` `\*[FR]`'. If an absolute footnote line length is required, recall that arithmetic expressions in `roff` input are evaluated strictly from left to right, with no operator precedence (parentheses are honored).

```
.ds FR 0+3i \" Set footnote line length to 3 inches.
```

[5] "Portable Document Format Publishing with GNU Troff", `pdfmark.ms` in the `groff` distribution, uses this technique.

[6] Unix Version 7 `ms`, its descendants, and GNU `ms` prior to `groff` version 1.23.0

[7] You could reset it after each call to `.1C`, `.2C`, or `.MC`.

#### 4.6.5.11 Language and localization

`groff ms` provides several strings that you can customize for your own purposes, or redefine to adapt the macro package to languages other than English. It is already localized for Czech, German, French, Italian, and Swedish. Load the desired localization macro package after `ms`; see the *groff_tmac*(5) man page.

```
$ groff -ms -mfr bienvenue.ms
```

The following strings are available.

`\*[REFERENCES]` [String]
Contains the string printed at the beginning of a references (bibliography) page produced with GNU *refer*(1). The default is '`References`'.

`\*[ABSTRACT]` [String]
Contains the string printed at the beginning of the abstract. The default is '`\f[I]ABSTRACT\f[]`'; it includes font selection escape sequences to set the word in italics.

`\*[TOC]` [String]
Contains the string printed at the beginning of the table of contents. The default is '`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]
Contain the full names of the calendar months. The defaults are in English: '`January`', '`February`', and so on.

### 4.6.6 Page layout

`ms`'s default page layout arranges text in a single column with the page number between hyphens centered in a header on each page except the first, and produces no footers. You can customize this arrangement.

#### 4.6.6.1 Headers and footers

There are multiple ways to produce headers and footers. One is to define the strings `LH`, `CH`, and `RH` to set the left, center, and right headers, respectively; and `LF`, `CF`, and `RF` to set the left, center, and right footers. This approach suffices for documents that do not distinguish odd- and even-numbered pages.

Another method is to call macros that set headers or footers for odd- or even-numbered pages. Each such macro takes a delimited argument separating the left, center, and right header or footer texts from each other. You can replace the neutral apostrophes (`'`) shown below with any character not appearing in the header or footer text. These macros are Berkeley extensions.

`.OH '`*left*`'`*center*`'`*right*`'` [Macro]
`.EH '`*left*`'`*center*`'`*right*`'` [Macro]
`.OF '`*left*`'`*center*`'`*right*`'` [Macro]
`.EF '`*left*`'`*center*`'`*right*`'` [Macro]
The `OH` and `EH` macros define headers for odd- (recto) and even-numbered (verso) pages, respectively; the `OF` and `EF` macros define footers for them.

By default, `ms` places no header on any page numbered "1" (regardless of its number format).

`.P1` [Macro]
Typeset the header even on page 1. To be effective, this macro must be called before the header trap is sprung on any page numbered "1"; in practice, unless your page numbering is unusual, this means that you should call it early, before `TL` or any heading or paragraphing macro. This is a Berkeley extension.

For even greater flexibility, `ms` is designed to permit the redefinition of the macros that are called when the `groff` traps that ordinarily cause the headers and footers to be output are sprung. `PT` ("page trap") is called by `ms` when the header is to be written, and `BT` ("bottom trap") when the footer is to be. The page location trap that `ms` sets up to format the header also calls the (normally undefined) `HD` macro after `PT`; you can define `HD` if you need additional processing after setting the header (for example, to draw a line below it). The `HD` hook is a Berkeley extension. Any such macros you (re)define must implement any desired specialization for odd-, even-, or first numbered pages.

#### 4.6.6.2 Tab stops

Use the `ta` request to define tab stops as needed. See Section 5.12 [Tabs and Fields], page 116.

`.TA` [Macro]
Reset the tab stops to the `ms` default (every 5 ens). Redefine this macro to create a different set of default tab stops.

### 4.6.6.3 Margins

Control margins using the registers summarized in "Margin settings" in Section 4.6.3 [ms Document Control Settings], page 30, above. There is no setting for the right margin; the combination of page offset `\n[PO]` and line length `\n[LL]` determines it.

### 4.6.6.4 Multiple columns

`ms` can set text in as many columns as reasonably fit on the page. The following macros force a page break if a multi-column layout is active when they are called. The `MINGW` register stores the default minimum gutter width; it is a GNU extension. When multiple columns are in use, keeps and the `HORPHANS` and `PORPHANS` registers work with respect to column breaks instead of page breaks.

`.1C` [Macro]
Arrange page text in a single column (the default).

`.2C` [Macro]
Arrange page text in two columns.

`.MC` [*column-width* [*gutter-width*]] [Macro]
Arrange page text in multiple columns. If you specify no arguments, it is equivalent to the `2C` macro. Otherwise, *column-width* is the width of each column and *gutter-width* is the minimum distance between columns.

### 4.6.6.5 Creating a table of contents

Because `roff` formatters process their input in a single pass, material on page 50, for example, cannot influence what appears on page 1—this poses a challenge for a table of contents at its traditional location in front matter, if you wish to avoid manually maintaining it. `ms` enables the collection of material to be presented in the table of contents as it appears, saving its page number along with it, and then emitting the collected contents on demand toward the end of the document. The table of contents can then be resequenced to its desired location as part of post-processing—with a PDF page relocation tool, or by physically rearranging the pages of a printed document, depending on the output format and circumstances.

Define an entry to appear in the table of contents by bracketing its text between calls to the `XS` and `XE` macros. A typical application is to call them immediately after `NH` or `SH` and repeat the heading text within them. The `XA` macro, used within '`.XS`'/'`.XE`' pairs, supplements an entry—for instance, when it requires multiple output lines, whether because a heading is too long to fit or because style dictates that page numbers not be repeated. You may

wish to indent the text thus wrapped to correspond to its heading depth; this can be done in the entry text by prefixing it with tabs or horizontal motion escape sequences, or by providing a second argument to the `XA` macro. `XS` and `XA` automatically associate the page number where they are called with the text following them, but they accept arguments to override this behavior. At the end of the document, call `TC` or `PX` to emit the table of contents; `TC` resets the page number to '`i`' (Roman numeral one), and then calls `PX`. All of these macros are Berkeley extensions.

`.XS` [*page-number*] [Macro]
`.XA` [*page-number* [*indentation*]] [Macro]
`.XE` [Macro]
Begin, supplement, and end a table of contents entry. Each entry is associated with *page-number* (otherwise the current page number); a *page-number* of '`no`' prevents a leader and page number from being emitted for that entry. Use of `XA` within `XS`/`XE` is optional; it can be repeated. If *indentation* is present, a supplemental entry is indented by that amount; ens are assumed if no unit is indicated. Text on input lines between `XS` and `XE` is stored for later recall by `PX`.

`.PX` [`no`] [Macro]
Switch to single-column layout. Unless `no` is specified, center and interpolate the `TOC` string in bold and two points larger than the body text. Emit the table of contents entries.

`.TC` [`no`] [Macro]
Set the page number to 1, the page number format to lowercase Roman numerals, and call `PX` (with a `no` argument, if present).

Here's an example of typical `ms` table of contents preparation. We employ horizontal escape sequences `\h` to indent the entries by sectioning depth.

```
.NH 1
Introduction
.XS
Introduction
.XE
...
.NH 2
Methodology
.XS
\h'2n'Methodology
.XA
\h'4n'Fassbinder's Approach
\h'4n'Kahiu's Approach
.XE
...
.NH 1
Findings
.XS
Findings
.XE
...
.TC
```

The remaining features in this subsubsection are GNU extensions. `groff ms` obviates the need to repeat heading text after `XS` calls. Call `XN` and `XH` after `NH` and `SH`, respectively.

`.XN` *heading-text* [Macro]
`.XH` *depth heading-text* [Macro]
Format *heading-text* and create a corresponding table of contents entry. `XN` computes the indentation from the depth of the preceding `NH` call; `XH` requires a *depth* argument to do so.

`groff ms` encourages customization of table of contents entry production.

`.XN-REPLACEMENT` *heading-text* [Macro]
`.XH-REPLACEMENT` *depth heading-text* [Macro]
These hook macros implement `XN` and `XH`, respectively. They call `XN-INIT` and pass their *heading-text* arguments to `XH-UPDATE-TOC`.

`.XN-INIT` [Macro]
`.XH-UPDATE-TOC` *depth heading-text* [Macro]
The `XN-INIT` hook macro does nothing by default. `XH-UPDATE-TOC` brackets *heading-text* with `XS` and `XE` calls, indenting it by 2 ens per level of *depth* beyond the first.

We could therefore produce a table of contents similar to that in the previous example with fewer macro calls. (The difference is that this input follows the "Approach" entries with leaders and page numbers.)

```
.NH 1
.XN Introduction
...
.NH 2
.XN Methodology
.XH 3 "Fassbinder's Approach"
.XH 3 "Kahiu's Approach"
...
.NH 1
.XN Findings
...
```

To get the section number of the numbered headings into the table of contents entries, we might define `XN-REPLACEMENT` as follows. (We obtain the heading depth from `groff ms`'s internal register `nh*hl`.)

```
.de XN-REPLACEMENT
.XN-INIT
.XH-UPDATE-TOC \\n[nh*hl] \\$@
\&\\*[SN] \\$*
..
```

You can change the style of the leader that bridges each table of contents entry with its page number; define the `TC-LEADER` special character by using the `char` request. A typical leader combines the dot glyph '`.`' with a horizontal motion escape sequence to spread the dots. The width of the page number field is stored in the `TC-MARGIN` register.

### 4.6.7 Differences from AT&T ms

The `groff ms` macros are an independent reimplementation, using no AT&T code. Since they take advantage of the extended features of `groff`, they cannot be used with AT&T `troff`. `groff ms` supports several features described above as Berkeley and Tenth Edition Research Unix extensions, and adds several of its own.

- The internals of `groff ms` differ from the internals of AT&T `ms`. Documents that depend upon implementation details of AT&T `ms` may not format properly with `groff ms`. Such details include macros whose function was not documented in the AT&T `ms` manual.[8]

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

- The error-handling policy of `groff ms` is to detect and report errors, rather than silently to ignore them.
- Tenth Edition Research Unix supported `P1`/`P2` macros to bracket code examples; `groff ms` does not.
- `groff ms` does not work in GNU `troff`'s AT&T compatibility mode. If loaded when that mode is enabled, it aborts processing with a diagnostic message.
- Multiple line spacing is not supported. Use a larger vertical spacing instead.
- `groff ms` uses the same header and footer defaults in both `nroff` and `troff` modes as AT&T `ms` does in `troff` mode; AT&T's default in `nroff` mode is to put the date, in U.S. traditional format (e.g., "January 1, 2021"), in the center footer (the `CF` string).
- Many `groff ms` macros, including those for paragraphs, headings, and displays, cause a reset of paragraph rendering parameters, and may change the indentation; they do so not by incrementing or decrementing it, but by setting it absolutely. This can cause problems for documents that define additional macros of their own that try to manipulate indentation. Use the `ms` `RS` and `RE` macros instead of the `in` request.
- AT&T `ms` interpreted the values of the registers `PS` and `VS` in points, and did not support the use of scaling units with them. `groff ms` interprets values of the registers `PS`, `VS`, `FPS`, and `FVS` equal to or larger than 1,000 (one thousand) as decimal fractions multiplied by 1,000.[9] This threshold makes use of a scaling unit with these parameters practical for high-resolution devices while preserving backward compatibility. It also permits expression of non-integral type sizes. For example, '`groff -rPS=10.5p`' at the shell prompt is equivalent to placing '`.nr PS 10.5p`' at the beginning of the document.
- AT&T `ms`'s `AU` macro supported arguments used with some document types; `groff ms` does not.
- Right-aligned displays are available. The AT&T `ms` manual observes that "it is tempting to assume that '`.DS R`' will right adjust lines, but it doesn't work". In `groff ms`, it does.
- To make `groff ms` use the default page offset (which also specifies the left margin), the `PO` register must stay undefined until the first `ms` macro is called.

  This implies that '`\n[PO]`' should not be used early in the document, unless it is changed also: accessing an undefined register automatically defines it.

[9] Register values are converted to and stored as basic units. See Section 5.3 [Measurements], page 77.

- `groff ms` supports the `PN` register, but it is not necessary; you can access the page number via the usual `%` register and invoke the `af` request to assign a different format to it if desired.[10]
- The AT&T `ms` manual documents registers `CW` and `GW` as setting the default column width and "intercolumn gap", respectively, and which applied when `MC` was called with fewer than two arguments. `groff ms` instead treats `MC` without arguments as synonymous with `2C`; there is thus no occasion for a default column width register. Further, the `MINGW` register and the second argument to `MC` specify a *minimum* space between columns, not the fixed gutter width of AT&T `ms`.
- The AT&T `ms` manual did not document the `QI` register; Berkeley and `groff ms` do.

`\n[GS]` [Register]

The register `GS` is set to 1 by the `groff ms` macros, but is not used by the AT&T `ms` package. Documents that need to determine whether they are being formatted with `groff ms` or another implementation should test this register.

#### 4.6.7.1 Unix Version 7 `ms` macros not implemented by `groff ms`

Several macros described in the Unix Version 7 `ms` documentation are unimplemented by `groff ms` because they are specific to the requirements of documents produced internally by Bell Laboratories, some of which also require a glyph for the Bell System logo that `groff` does not support. These macros implemented several document type formats (`EG`, `IM`, `MF`, `MR`, `TM`, `TR`), were meaningful only in conjunction with the use of certain document types (`AT`, `CS`, `CT`, `OK`, `SG`), stored the postal addresses of Bell Labs sites (`HO`, `IH`, `MH`, `PY`, `WH`), or lacked a stable definition over time (`UX`). To compatibly render historical `ms` documents using these macros, we advise your documents to invoke the `rm` request to remove any such macros it uses and then define replacements with an authentically typeset original at hand.[11] For informal purposes, a simple definition of `UX` should maintain the readability of the document's substance.

```
.rm UX
.ds UX Unix\"
```

[10] If you redefine the `ms PT` macro and desire special treatment of certain page numbers (like '`1`'), you may need to handle a non-Arabic page number format, as `groff ms`'s `PT` does; see the macro package source. `groff ms` aliases the `PN` register to `%`.

[11] The removal beforehand is necessary because `groff ms` aliases these macros to a diagnostic macro, and you want to redefine the aliased name, not its target.

### 4.6.8 Legacy Features

`groff ms` retains some legacy features solely to support formatting of historical documents; contemporary ones should not use them because they can render poorly. See the *groff_char*(7) man page.

#### AT&T accent mark strings

AT&T `ms` defined accent mark strings as follows.

`\*[']` [String]
Apply acute accent to subsequent glyph.

`\*[`]` [String]
Apply grave accent to subsequent glyph.

`\*[:]` [String]
Apply dieresis (umlaut) to subsequent glyph.

`\*[^]` [String]
Apply circumflex accent to subsequent glyph.

`\*[~]` [String]
Apply tilde accent to subsequent glyph.

`\*[C]` [String]
Apply caron to subsequent glyph.

`\*[,]` [String]
Apply cedilla to subsequent glyph.

#### Berkeley accent mark and glyph strings

Berkeley `ms` offered an `AM` macro; calling it redefined the AT&T accent mark strings (except for '`\*C`'), applied them to the *preceding* glyph, and defined additional strings, some for spacing glyphs.

`.AM` [Macro]
Enable alternative accent mark and glyph-producing strings.

`\*[']` [String]
Apply acute accent to preceding glyph.

`\*[`]` [String]
Apply grave accent to preceding glyph.

`\*[:]` [String]
Apply dieresis (umlaut) to preceding glyph.

`\*[^]` [String]
Apply circumflex accent to preceding glyph.

`\*[~]` [String]
Apply tilde accent to preceding glyph.

`\*[,]` [String]
Apply cedilla to preceding glyph.

`\*[/]` [String]
Apply stroke (slash) to preceding glyph.

`\*[v]` [String]
Apply caron to preceding glyph.

`\*[_]` [String]
Apply macron to preceding glyph.

`\*[.]` [String]
Apply underdot to preceding glyph.

`\*[o]` [String]
Apply ring accent to preceding glyph.

`\*[?]` [String]
Interpolate inverted question mark.

`\*[!]` [String]
Interpolate inverted exclamation mark.

`\*[8]` [String]
Interpolate small letter sharp s.

`\*[q]` [String]
Interpolate small letter o with hook accent (ogonek).

`\*[3]` [String]
Interpolate small letter yogh.

`\*[d-]` [String]
Interpolate small letter eth.

`\*[D-]` [String]
Interpolate capital letter eth.

`\*[th]` [String]
Interpolate small letter thorn.

`\*[Th]` [String]
Interpolate capital letter thorn.

`\*[ae]` [String]
Interpolate small æ ligature.

`\*[Ae]` [String]
Interpolate capital Æ ligature.

`\*[oe]` [String]
Interpolate small oe ligature.

`\*[OE]` [String]
Interpolate capital OE ligature.

### 4.6.9 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 GNU troff 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 drafting a document of concern with details like line length, hyphenation breaking, and the achievement of straight margins. 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.

`roff` input files contain text interspersed with instructions to control the formatter. Even in the absence of such instructions, GNU `troff` still processes its input in several ways, by filling, hyphenating, breaking, and adjusting it, and supplementing it with inter-sentence space.

### 5.1.1 Filling

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

GNU `troff` interprets plain text files employing the Unix line-ending convention. It reads input a character at a time, collecting words as it goes, and fits as many words together on an output line as it can—this is known as *filling*. To GNU `troff`, a *word* is any sequence of one or more characters that aren't spaces, tabs, or newlines. The exceptions separate words.[2] To disable filling, see Section 5.9 [Manipulating Filling and Adjustment], page 101.

```
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] See Section 6.2 [Device and Font Description Files], page 244.

[2] There are also *escape sequences* which can function as word characters, word separators, or neither—the last simply have no effect on GNU `troff`'s idea of whether an input character 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`; it attempts to detect the boundaries between sentences, and supplies additional inter-sentence space between them.

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

GNU `troff` does this by flagging certain characters (normally '`!`', '`?`', and '`.`') as potentially ending a sentence. When GNU `troff` encounters one of these *end-of-sentence characters* at the end of a line, or one of them is followed by two spaces on the same input line, it appends an inter-word 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 instructions to GNU `troff` instead of being used to construct 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 dummy character 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.
```

Adding text caused our input to wrap; now, we don't need `\&` after '`T.`' but we do after '`P.`'. Consistent use of the escape sequence ensures that potential sentence boundaries are robust to editing activities. Further advice along these lines will follow in Section 5.1.10 [Input Conventions], page 73.

Normally, the occurrence of a visible non-end-of-sentence character (as opposed to a space or tab) immediately after an end-of-sentence character cancels detection of the end of a sentence. 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 occurrence of an end-of-sentence character. That is, GNU `troff` does not cancel end-of-sentence detection 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).[5]

```
\[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 5.19.4 [Using Symbols], page 137. To change the additional inter-sentence space amount—even to remove it entirely—see Section 5.9 [Manipulating Filling and Adjustment], page 101.

### 5.1.3 Hyphenation

When an output line is nearly full, it is uncommon for the next word collected from the input to exactly fill it—typically, there is room left over only 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) with its remainder on the next is *hyphenation*. Hyphenation points can be manually specified; GNU `troff` also uses a hyphenation algorithm and language-specific pattern files (based on those used in TEX) to decide which words can be hyphenated and where.

[5] The mnemonics for the special characters shown here are "dagger", "double dagger", "right (double) quote", and "closing (single) quote". See the *groff_char*(7) man page.

Hyphenation does not always occur even when the hyphenation rules for a word allow it; it can be disabled, and when not disabled there are several parameters that can prevent it in certain circumstances. See Section 5.10 [Manipulating Hyphenation], page 107.

### 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. When the formatter is filling text, it introduces breaks automatically to keep output lines from exceeding the configured line length. 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 usually does not happen with natural language text unless the output line length has been manipulated to be extremely short, but it can with specialized text like program source code. We can use `perl` at the shell prompt to contrive an example of failure to break the line. We also employ the `-z` option to suppress normal output.

```
$ perl -e 'print "#" x 80, "\n";' | nroff -z
     error  warning: cannot 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 5.10 [Manipulating Hyphenation], page 107.

What if the document author wants to stop filling lines temporarily, for instance to start a new paragraph? There are several solutions. A blank input line not only causes a break, but by default it also outputs a one-line vertical space (effectively a blank output line). This behavior can be modified; see Section 5.28.3 [Blank Line Traps], page 195. Macro packages may discourage or disable the blank line method of paragraphing in favor of their own macros.

A line that begins with one or more spaces causes a break. The spaces are output at the beginning of the next line without being *adjusted* (see below); however, this behavior can be modified (see Section 5.28.4 [Leading Space Traps], page 195). Again, macro packages may provide other methods of producing indented paragraphs. Trailing spaces on text lines are discarded.[6]

What if the file ends before enough words have been collected to fill an output line? Or the output line is exactly full but not yet broken, and there is no more input? GNU `troff` interprets the end of input as a break. Certain requests also cause breaks, implicitly or explicitly. This is discussed in Section 5.9 [Manipulating Filling and Adjustment], page 101.

[6] "Text lines" are defined in Section 5.1.7 [Requests and Macros], page 69.

### 5.1.5 Adjustment

After GNU troff performs an automatic break, it may then *adjust* the line, widening inter-word spaces until the text reaches the right margin. Extra spaces between words are preserved. Leading and trailing spaces are handled as noted above. Text can be aligned to the left or right margin only, or centered; see Section 5.9 [Manipulating Filling and Adjustment], page 101.

### 5.1.6 Tabs and Leaders

GNU troff translates input horizontal tab characters ("tabs") and Control+A characters ("leaders") into movements to the next tab stop. Tabs simply move to the next tab stop; leaders place enough periods to fill the space. Tab stops are by default located every half inch measured from the drawing position corresponding to the beginning of the input line; see Section 5.2 [Page Geometry], page 76. Tabs and leaders do not cause breaks and therefore do not interrupt filling. Below, we use arrows → and bullets • to indicate input tabs and leaders, respectively.

```
1
→ 2 → 3 • 4
→ • 5
⇒ 1          2       3.......4          ........5
```

Tabs and leaders lend themselves to table construction.[7] The tab and leader glyphs can be configured, and further facilities for sophisticated table composition are available; see Section 5.12 [Tabs and Fields], page 116. There are many details to track when using such low-level features, so most users turn to the *tbl*(1) preprocessor for table construction.

### 5.1.7 Requests and Macros

We have now encountered almost all of the syntax there is in the roff language, with an exception already noted in passing. A *request* is an instruction to the formatter that occurs after a *control character*, which is recognized at the beginning of an input line. The regular control character is a dot (.). Its counterpart, the *no-break control character*, a neutral apostrophe ('), suppresses the break that is implied by some requests. These characters were chosen because it is uncommon for lines of text in natural languages to begin with them. If you require a formatted period or apostrophe (closing single quotation mark) where GNU troff is expecting a control character, prefix the dot or neutral apostrophe with the dummy character escape sequence, '\&'.

An input line beginning with a control character is called a *control line*. Every line of input that is not a control line is a *text line*.[8]

---

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

[8] The \RET escape sequence can alter how an input line is classified; see Section 5.16 [Line Continuation], page 127.

Requests often take *arguments*, words (separated from the request name and each other by spaces) that specify details of the action GNU `troff` is expected to perform. If a request is meaningless without arguments, it is typically ignored.

GNU `troff`'s requests and escape sequences comprise the control language of the formatter. Of key importance are the requests that define macros. Macros are invoked like requests, enabling the request repertoire to be extended or overridden.[9]

A *macro* can be thought of as an abbreviation you can define for a collection of control and text lines. When the macro is *called* by giving its name after a control character, it is replaced with what it stands for. The process of textual replacement is known as *interpolation*.[10] 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.[11]

```
.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 ends the macro definition. This is a default; you can specify your own terminator for the macro definition as the second argument to the `de` request.

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

In fact, the ending marker is itself the name of a macro that will be called if it is defined by that time (or the name of a request to be invoked).

```
.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?

---

[9] Argument handling in macros is more flexible but also more complex. See Section 5.6.3 [Calling Macros], page 88.

[10] Some escape sequences undergo interpolation as well.

[11] GNU `troff` offers additional ones. See Section 5.24 [Writing Macros], page 168.

Let us consider a more elaborate example.

```
.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 above document started with a series of control lines. Three macros were defined, with a `de` request declaring each 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 5.22 [Strings], page 157) 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 5.24.2 [Copy Mode], page 174). While a macro is being defined (or appended to), requests are not interpreted and macros not interpolated, whereas some commonly used escape sequences *are* interpreted. `roff` systems also support recursive macro calls, as long as you have a way to break the recursion (see Section 5.23 [Conditionals and Loops], page 161). Maintainable `roff` documents tend to arrange macro definitions to minimize forward references.

### 5.1.8 Macro Packages

Macro definitions can be collected into *macro files*, `roff` input files designed to produce no output themselves but instead ease the preparation of other `roff` documents. There is no syntactical difference between a macro file and any other `roff` document; only its purpose distinguishes it. When a macro file is installed at a standard location and suitable for use by a general audience, it is often termed a *macro package*.[12] Macro packages can be loaded by supplying the `-m` option to GNU `troff` or a `groff` front end. A document wishing to use a macro package can load it with the `mso` ("macro source") request.

### 5.1.9 Input Encodings

The `groff` command's `-k` option calls the `preconv` preprocessor to perform input character encoding conversions. Input to the GNU `troff` formatter itself, 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 loaded at startup.

`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 startup.

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

[12] Macro files and packages frequently define registers and strings as well.

valid ISO Latin-1 document; the standards are interchangeable in their first 128 code points.[13]

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.

latin9 ISO Latin-9 is a successor to Latin-1. Its main difference from Latin-1 is that Latin-9 contains the Euro sign. 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 characters from an input encoding may not be available with a particular output driver, or their glyphs may not have representation in the font used. For terminal devices, fallbacks are defined, like 'EUR' for the Euro sign and '(C)' for the copyright sign. For typesetter devices, you may need to "mount" fonts that support glyphs required by the document. See Section 5.19.3 [Font Positions], page 135.

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. Because standard PostScript fonts contain the other glyphs from Latin-5 and Latin-9 that Latin-1 lacks, these encodings are supported for the ps and pdf output devices as groff ships, while Latin-2 is not.

Unicode supports characters from all other input encodings; the utf8 output driver for terminals therefore does as well. The DVI output driver supports the Latin-2 and Latin-9 encodings if the command-line option -mec is used as well.[14]

### 5.1.10 Input Conventions

Since GNU troff fills text automatically, it is common practice in the roff language to avoid visual composition of text in input files: the esthetic appeal of the formatted output is what matters. Therefore, roff input should be arranged such that it is easy for authors and maintainers to compose and

[13] 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.

[14] The DVI output device defaults to using the Computer Modern (CM) fonts; ec.tmac loads the EC fonts instead, which provide Euro '\[Eu]' and per mille '\[%0]' glyphs.

develop the document, understand the syntax of `roff` requests, macro calls, and preprocessor languages used, and predict the behavior of the formatter. Several traditions have accrued in service of these goals.

- Follow sentence endings in the input with newlines to ease their recognition (see Section 5.1.2 [Sentences], page 66). It is frequently convenient to end text lines after colons and semicolons as well, as these typically precede independent clauses. Consider doing so after commas; they often occur in lists that become easy to scan when itemized by line, or constitute supplements to the sentence that are added, deleted, or updated to clarify it. Parenthetical and quoted phrases are also good candidates for placement on text lines by themselves.
- Set your text editor's line length to 72 characters or fewer.[15] This limit, combined with the previous item of advice, makes it less common that an input line will wrap in your text editor, and thus will help you perceive excessively long constructions in your text. Recall that natural languages originate in speech, not writing, and that punctuation is correlated with pauses for breathing and changes in prosody.
- Use `\&` after '`!`', '`?`', and '`.`' if they are followed by space, tab, or newline characters and don't end a sentence.
- In filled text lines, use `\&` before '`.`' and '`'`' if they are preceded by space, so that reflowing the input doesn't turn them into control lines.
- Do not use spaces to perform indentation or align columns of a table. Leading spaces are reliable when text is not being filled.
- 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—a control character followed immediately by a newline—to visually manage separation of material in input files. Many of the `groff` project's own documents use an empty request between sentences, after macro definitions, and where a break is expected, and two empty requests between paragraphs or other requests or macro calls that will introduce vertical space into the document.

  You can combine the empty request with the comment escape sequence to include whole-line comments in your document, and even "comment out" sections of it.

We conclude this section with an example sufficiently long to illustrate most of the above suggestions in practice. For the purpose of fitting the example between the margins of this manual with the font used for its typeset version, we have shortened the input line length to 56 columns. As before, an arrow → indicates a tab character.

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

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

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

## 5.2 Page Geometry

`roff` systems format text under certain assumptions about the size of the output medium, or page. For the formatter to correctly break a line it is filling, it must know the line length, which it derives from the page width (see Section 5.15 [Line Layout], page 124). For it to decide whether to write an output line to the current page or wait until the next one, it must know the page length (see Section 5.17 [Page Layout], page 128).

A device's *resolution* converts practical units like inches or centimeters to *basic units*, a convenient length measure for the output device or file format. The formatter and output driver use basic units to reckon page measurements. The device description file defines its resolution and page dimensions (see Section 6.2.1 [DESC File Format], page 244).

A *page* is a two-dimensional structure upon which a `roff` system imposes a rectangular coordinate system with its upper left corner as the origin. Coordinate values are in basic units and increase down and to the right. Useful ones are therefore always positive and within numeric ranges corresponding to the page boundaries.

While the formatter (and, later, output driver) is processing a page, it keeps track of its *drawing position*, which is the location at which the next glyph will be written, from which the next motion will be measured, or where a geometric primitive will commence rendering. Notionally, glyphs are drawn from the text baseline upward and to the right.[16] The *text baseline* is a (usually invisible) line upon which the glyphs of a typeface are aligned. A glyph therefore "starts" at its bottom-left corner. If drawn at the origin, a typical letter glyph would lie partially or wholly off the page, depending on whether, like "g", it features a decender below the baseline.

Such a situation is nearly always undesirable. It is furthermore conventional not to write or draw at the extreme edges of the page. Therefore the initial drawing position of a `roff` formatter is not at the origin, but below and to the right of it. This rightward shift from the left edge is known as the *page offset*.[17] The downward shift leaves room for a text output line.

Text is arranged on a one-dimensional lattice of text baselines from the top to the bottom of the page. *Vertical spacing* is the distance between adjacent text baselines. Typographic tradition sets this quantity to 120% of the type size. The initial drawing position is one unit of vertical spacing below the page top. Typographers term this unit a *vee*.

Vertical spacing has an impact on page-breaking decisions. Generally, when a break occurs, the formatter moves the drawing position to the next text baseline automatically. If the formatter were already writing to the last line that would fit on the page, advancing by one vee would place the next text baseline off the page. Rather than let that happen, `roff` formatters

[16] `groff` does not yet support right-to-left scripts.

[17] `groff`'s terminal output devices have page offsets of zero.

instruct the output driver to eject the page, start a new one, and again set the drawing position to one vee below the page top; this is a *page break*.

When the last line of input text corresponds to the last output line that fits on the page, the break caused by the end of input will also break the page, producing a useless blank one. Macro packages keep users from having to confront this difficulty by setting "traps" (see Section 5.28 [Traps], page 187); moreover, all but the simplest page layouts tend to have headers and footers, or at least bear vertical margins larger than one vee.

## 5.3 Measurements

The formatter sometimes requires the input of numeric parameters to specify measurements. These are specified as integers or decimal fractions with an optional *scaling unit* suffixed. A scaling unit is a letter that immediately follows the last digit of a number. Digits after the decimal point are optional. Measurement expressions include '`10.5p`', '`11i`', and '`3.c`'.

Measurements are scaled by the scaling unit and stored internally (with any fractional part discarded) in basic units. The device resolution can therefore be obtained by storing a value of '`1i`' to a register. The only constraint on the basic unit is that it is at least as small as any other unit.

`u` Basic unit.

`i` Inch; defined as 2.54 centimeters.

`c` Centimeter; a centimeter is about 0.3937 inches.

`p` Point; a typesetter's unit used for measuring type size. There are 72 points to an inch.

`P` Pica; another typesetter's unit. There are 6 picas to an inch and 12 points to a pica.

`s`
`z` See Section 5.20.3 [Using Fractional Type Sizes], page 154, for a discussion of these units.

`f` GNU `troff` defines this unit to scale decimal fractions in the interval [0, 1] to 16-bit unsigned integers. It multiplies a quantity by 65,536. See Section 5.21 [Colors], page 155, for usage.

The magnitudes of other scaling units depend on the text formatting parameters in effect. These are useful when specifying measurements that need to scale with the typeface or vertical spacing.

`m` Em; an em is equal to the current type size in points. It is named thus because it is approximately the width of the letter '`M`'.

`n` En; an en is one-half em.

`v` Vee; recall Section 5.2 [Page Geometry], page 76.

`M` Hundredth of an em.

### 5.3.1 Motion Quanta

An output device's basic unit `u` is not necessarily its smallest addressable length; `u` can be smaller to avoid problems with integer roundoff. The minimum distances that a device can work with in the horizontal and vertical directions are termed its *motion quanta*. Measurements are rounded to applicable motion quanta. Half-quantum fractions round toward zero.

`\n[.H]` [Register]
`\n[.V]` [Register]

These read-only registers interpolate the horizontal and vertical motion quanta, respectively, of the output device in basic units.

For example, we might draw short baseline rules on a terminal device as follows. See Section 5.26 [Drawing Requests], page 182.

```
.tm \n[.H]
    error  24
.nf
\l'36u' 36u
\l'37u' 37u
    ⇒ _ 36u
    ⇒ __ 37u
```

### 5.3.2 Default Units

A general-purpose register (one created or updated with the `nr` request; see see Section 5.8 [Registers], page 94) is implicitly dimensionless, or reckoned in basic units if interpreted in a measurement context. But it is convenient for many requests and escape sequences to infer a scaling unit for an argument if none is specified. An explicit scaling unit (not after a closing parenthesis) can override an undesirable default. Effectively, the default unit is suffixed to the expression if a scaling unit is not already present. GNU `troff`'s use of integer arithmetic should also be kept in mind (see Section 5.4 [Numeric Expressions], page 79).

The `ll` request interprets its argument in ems by default. Consider several attempts to set a line length of 3.5 inches when the type size is 10 points on a terminal device with a resolution of 240 basic units and horizontal motion quantum of 24. Some expressions become zero; the request clamps them to that quantum.

```
.ll 3.5i      \" 3.5i (= 840u)
.ll 7/2       \" 7u/2u -> 3u -> 3m -> 0, clamped to 24u
.ll (7 / 2)u  \" 7u/2u -> as above
.ll 7/2i      \" 7u/2i -> 7u/480u -> 0 -> as above
.ll 7i/2      \" 7i/2u -> 1680u/2m -> 1680u/24u -> 35u
.ll 7i/2u     \" 3.5i (= 840u)
```

The safest way to specify measurements is to attach a scaling unit. To multiply or divide by a dimensionless quantity, use '`u`' as its scaling unit.

## 5.4 Numeric Expressions

A *numeric expression* evaluates to an integer: it can be as simple as a literal '0' or it can be a complex sequence of register and string interpolations interleaved with measurements and operators.

GNU troff provides a set of mathematical and logical operators familiar to programmers—as well as some unusual ones—but supports only integer arithmetic.[18] The internal data type used for computing results is usually a 32-bit signed integer, which suffices to represent magnitudes within a range of $\pm 2$ billion.[19]

Arithmetic infix operators perform a function on the numeric expressions to their left and right; they are + (addition), - (subtraction), * (multiplication), / (truncating division), and % (modulus). *Truncating division* rounds to the integer nearer to zero, no matter how large the fractional portion. Overflow and division (or modulus) by zero are errors and abort evaluation of a numeric expression.

Arithmetic unary operators operate on the numeric expression to their right; they are - (negation) and + (assertion—for completeness; it does nothing). The unary minus must often be used with parentheses to avoid confusion with the decrementation operator, discussed below.

Observe the rounding behavior and effect of negative operands on the modulus and truncating division operators.

```
.nr T 199/100
.nr U 5/2
.nr V (-5)/2
.nr W 5/-2
.nr X 5%2
.nr Y (-5)%2
.nr Z 5%-2
T=\n[T] U=\n[U] V=\n[V] W=\n[W] X=\n[X] Y=\n[Y] Z=\n[Z]
    ⇒ T=1 U=2 V=-2 W=-2 X=1 Y=-1 Z=1
```

The sign of the modulus of operands of mixed signs is determined by the sign of the first. Division and modulus operators satisfy the following property: given a dividend $a$ and a divisor $b$, a quotient $q$ formed by '(a / b)' and a remainder $r$ by '(a % b)', then $qb + r = a$.

GNU troff's scaling operator, used with parentheses as (c;e), evaluates a numeric expression $e$ using $c$ as the default scaling unit. If $c$ is omitted, scaling units are ignored in the evaluation of $e$. This operator can save typing by avoiding the attachment of scaling units to every operand out of caution. Your macros can select a sensible default unit in case the user neglects to supply one.

---

[18] Provision is made for interpreting and reporting decimal fractions in certain cases.

[19] If that's not enough, see the *groff_tmac*(5) man page for the 62bit.tmac macro package.

```
.\" Indent by amount given in first argument; assume ens.
.de Indent
.  in (n;\\$1)
..
```

Without the scaling operator, the foregoing macro would, if called with a unitless argument, cause indentation by the `in` request's default scaling unit (ems). The result would be twice as much indentation as expected.

GNU `troff` also provides a pair of operators to compute the extrema of two operands: `>?` (maximum) and `<?` (minimum).

```
.nr slots 5
.nr candidates 3
.nr salaries (\n[slots] <? \n[candidates])
Looks like we'll end up paying \n[salaries] salaries.
    ⇒ Looks like we'll end up paying 3 salaries.
```

Comparison operators comprise `<` (less than), `>` (greater than), `<=` (less than or equal), `>=` (greater than or equal), and `=` (equal). `==` is a synonym for `=`. When evaluated, a comparison is replaced with '`0`' if it is false and '`1`' if true. In the `roff` language, positive values are true, others false.

We can operate on truth values with the logical operators `&` (logical conjunction or "and") and `:` (logical disjunction or "or"). They evaluate as comparison operators do.

A logical complementation ("not") operator, `!`, works only within `if`, `ie`, and `while` requests. Furthermore, `!` is recognized only at the beginning of a numeric expression not contained by another numeric expression. In other words, it must be the "outermost" operator. Including it elsewhere in the expression produces a warning in the '`number`' category (see Section 5.37.1 [Warnings], page 222), and its expression evaluates false. This unfortunate limitation maintains compatibility with AT&T `troff`. You can test a numeric expression for falsity by comparing it to a false value.

```
.nr X 1
.nr Y 0
.\" This does not work as expected.
.if (\n[X])&(!\n[Y]) .nop A: X is true, Y is false
.
.\" Use this construct instead.
.if (\n[X])&(\n[Y]<=0) .nop B: X is true, Y is false
    error  warning: expected numeric expression, got '!'
    ⇒ B: X is true, Y is false
```

The `roff` language has no operator precedence: expressions are evaluated strictly from left to right, in contrast to schoolhouse arithmetic. Use parentheses `( )` to impose a desired precedence upon subexpressions.

```
.nr X 3+5*4
.nr Y (3+5)*4
.nr Z 3+(5*4)
X=\n[X] Y=\n[Y] Z=\n[Z]
    ⇒ X=32 Y=32 Z=23
```

For many requests and escape sequences that cause motion on the page, the unary operators `+` and `-` work differently when leading a numeric expression. They then indicate a motion relative to the drawing position: positive is down in vertical contexts, right in horizontal ones.

`+` and `-` are also treated differently by the following requests and escape sequences: `bp`, `in`, `ll`, `lt`, `nm`, `nr`, `pl`, `pn`, `po`, `ps`, `pvs`, `rt`, `ti`, `\H`, `\R`, and `\s`. Here, leading plus and minus signs serve as incrementation and decrementation operators, respectively. To negate an expression, subtract it from zero or include the unary minus in parentheses with its argument. See Section 5.8.1 [Setting Registers], page 94, for examples.

A leading `|` operator indicates a motion relative not to the drawing position but to a boundary. For horizontal motions, the measurement specifies a distance relative to a drawing position corresponding to the beginning of the *input* line. By default, tab stops reckon movements in this way. Most escape sequences do not; `|` tells them to do so.

```
Mind the \h'1.2i'gap.
.br
Mind the \h'|1.2i'gap.
.br
Mind the
\h'|1.2i'gap.
    ⇒ Mind the             gap.
    ⇒ Mind the    gap.
    ⇒ Mind the             gap.
```

One use of this feature is to define macros whose scope is limited to the output they format.

```
.\" underline word $1 with trailing punctuation $2
.de Underline
.  nop \\$1\l'|0\[ul]'\\$2
..
Typographical emphasis is best used
.Underline sparingly .
```

In the above example, '`|0`' specifies a negative motion from the current position (at the end of the argument just emitted, `\$1`) to the beginning of the input line. Thus, the `\l` escape sequence in this case draws a line from right to left. A macro call occurs at the beginning of an input line;[20]

[20] Control structure syntax creates an exception to this rule, but is designed to remain useful: recalling our example, '`.if 1 .Underline this`' would underline only "this", precisely. See Section 5.23 [Conditionals and Loops], page 161.

if the | operator were omitted, then the underline would be drawn at zero distance from the current position, producing device-dependent, and likely undesirable, results. On the 'ps' output device, it underlines the period.

For vertical movements, the | operator specifies a distance from the first text baseline on the page or in the current diversion, using the current vertical spacing.

```
A
.br
B \Z'C'\v'|0'D
     ⇒ A D
     ⇒ B C
```

In the foregoing example, we've used the \Z escape sequence (see Section 5.25 [Page Motions], page 177) to restore the drawing position after formatting 'C', then moved vertically to the first text baseline on the page.

\B'*anything*' [Escape sequence]
Interpolate 1 if *anything* is a valid numeric expression, and 0 otherwise. The delimiter need not be a neutral apostrophe; see Section 5.6.5 [Delimiters], page 91.

You might use \B along with the `if` request to filter out invalid macro or string arguments. See Section 5.23 [Conditionals and Loops], page 161.

```
.\" Indent by amount given in first argument; assume ens.
.de Indent
.  if \B'\\$1' .in (n;\\$1)
..
```

A register interpolated as an operand in a numeric expression must have an Arabic format; luckily, this is the default. See Section 5.8.4 [Assigning Register Formats], page 98.

Because spaces separate arguments to requests, spaces are not allowed in numeric expressions unless the (sub)expression containing them is surrounded by parentheses. See Section 5.6.2 [Invoking Requests], page 86, and Section 5.23 [Conditionals and Loops], page 161.

```
.nf
.nr a 1+2 + 2+1
\na
     error  expected numeric expression, got a space
     ⇒ 3
.nr a 1+(2 + 2)+1
\na
     ⇒ 6
```

The `nr` request (see Section 5.8.1 [Setting Registers], page 94) expects its second and optional third arguments to be numeric expressions; a bare + does not qualify, so our first attempt got a warning.

## 5.5 Identifiers

GNU troff has rules for properly formed *identifiers*—labels for objects with syntactical importance, like registers, names (macros, strings, or diversions), typefaces, glyphs, colors, character classes, environments, and streams. An identifier consists of one or more characters excepting spaces, tabs, newlines, and invalid input characters.

Invalid input characters are a subset of control characters (from the sets "C0 Controls" and "C1 Controls" as Unicode describes them). When GNU troff encounters one in an identifier, it produces a warning in category 'input' (see Section 5.37.1 [Warnings], page 222). They are removed during interpretation: an identifier 'foo', followed by an invalid character and then 'bar', is processed as 'foobar'.

On a machine using the ISO 646, 8859, or 10646 character encodings, invalid input characters are 0x00, 0x08, 0x0B, 0x0D–0x1F, and 0x80–0x9F. On an EBCDIC host, they are 0x00–0x01, 0x08, 0x09, 0x0B, 0x0D–0x14, 0x17–0x1F, and 0x30–0x3F.[21] Some of these code points are used by GNU troff internally, making it non-trivial to extend the program to accept UTF-8 or other encodings that use characters from these ranges.[22]

The identifiers 'br', 'PP', 'end-list', 'ref*normal-print', '|', '@_', and '!"#$%'()*+,-./' are all valid. Discretion should be exercised to prevent confusion. Some care is required with identifiers starting with '(' or '['.

```
.nr x 9
.nr y 1
.nr (x 2
.nr [y 3
.nr sum1 (\n(x + \n[y])
     error  a space character is not allowed in an escape
     error    sequence parameter
A:2+3=\n[sum1]
.nr sum2 (\n((x + \n[[y])
B:2+3=\n[sum2]
.nr sum3 (\n[(x] + \n([y)
C:2+3=\n[sum3]
    ⇒ A:2+3=1 B:2+3=5 C:2+3=5
```

[21] Historically, control characters like ASCII STX, ETX, and BEL (Control+B, Control+C, and Control+G) have been observed in roff documents, particularly in macro packages employing them as delimiters with the output comparison operator to try to avoid collisions with the content of arbitrary user-supplied parameters (see Section 5.23.1 [Operators in Conditionals], page 161). We discourage this expedient; in GNU troff it is unnecessary (outside of compatibility mode) because delimited arguments are parsed at a different input level than the surrounding context. See Section 5.38 [Implementation Differences], page 224.

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

An identifier with a closing bracket (‘`]`’) in its name can't be accessed with bracket-form escape sequences that expect an identifier as a parameter. For example, ‘`\[foo]]`’ accesses the glyph ‘`foo`’, followed by ‘`]`’ in whatever the surrounding context is, whereas ‘`\C'foo]'`’ formats a glyph named ‘`foo]`’. Similarly, the identifier ‘`(`’ can't be interpolated *except* with bracket forms.

If you begin a macro, string, or diversion name with either of the characters ‘`[`’ or ‘`]`’, you foreclose use of the `grefer` preprocessor, which recognizes ‘`.[`’ and ‘`.]`’ as bibliographic reference delimiters.

`\A'`*anything*`'` [Escape sequence]

Interpolate 1 if *anything* is a valid identifier, and 0 otherwise. The delimiter need not be a neutral apostrophe; see Section 5.6.5 [Delimiters], page 91. Because invalid input characters are removed (see above), invalid identifiers are empty or contain spaces, tabs, or newlines.

You can employ `\A` to validate a macro argument before using it to construct another escape sequence or identifier.

```
.\" usage: .init-coordinate-pair name val1 val2
.\" Create a coordinate pair where name!x=val1 and
.\" name!y=val2.
.de init-coordinate-pair
.  if \A'\\$1' \{\
.    if \B'\\$2' .nr \\$1!x \\$2
.    if \B'\\$3' .nr \\$1!y \\$3
.  \}
..
.init-coordinate-pair center 5 10
The center is at (\n[center!x], \n[center!y]).
.init-coordinate-pair "poi→nt" trash garbage \" ignored
.init-coordinate-pair point trash garbage \" ignored
    ⇒ The center is at (5, 10).
```

In this example, we also validated the numeric arguments; the registers ‘`point!x`’ and ‘`point!y`’ remain undefined. See Section 5.4 [Numeric Expressions], page 79, for the `\B` escape sequence.

How GNU `troff` handles the interpretation of an undefined identifier depends on the context. There is no way to invoke an undefined request; such syntax is interpreted as a macro call instead. If the identifier is being interpreted as a string, macro, or diversion, GNU `troff` emits a warning in category ‘`mac`’, defines it as empty, and interpolates nothing. If the identifier is being interpreted as a register, GNU `troff` emits a warning in category ‘`reg`’, initializes it to zero, and interpolates that value. See Section 5.37.1 [Warnings], page 222, Section 5.8.2 [Interpolating Registers], page 96, and Section 5.22 [Strings], page 157. Attempting to use an undefined typeface, glyph, color, character class, environment, or stream generally provokes an error diagnostic.

Identifiers for requests, macros, strings, and diversions share one name space; special characters and character classes another. No other object types do.

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

The foregoing example shows that GNU troff reuses the identifier 'xxx', changing it from a macro to a diversion. No warning is emitted, and the previous contents of 'xxx' are lost.

## 5.6 Formatter Instructions

To support documents that require more than filling, automatic line breaking and hyphenation, adjustment, and supplemental inter-sentence space, the roff language offers two means of embedding instructions to the formatter.

One is a *request*, which begins with a control character and takes up the remainder of the input line. Requests often perform relatively large-scale operations such as setting the page length, breaking the line, or starting a new page. They also conduct internal operations like defining macros.

The other is an *escape sequence*, which begins with the escape character and can be embedded anywhere in the input, even in arguments to requests and other escape sequences. Escape sequences interpolate special characters, strings, or registers, and handle comparatively minor formatting tasks like sub- and superscripting.

Some operations, such as font selection and type size alteration, are available via both requests and escape sequences.

### 5.6.1 Control Characters

The mechanism of using roff's control characters to invoke requests and call macros was introduced in Section 5.1.7 [Requests and Macros], page 69. Control characters are recognized only at the beginning of an input line, or at the beginning of the branch of a control structure request; see Section 5.23 [Conditionals and Loops], page 161.

A few requests cause a break implicitly; use the no-break control character to prevent the break. Break suppression is its sole behavioral distinction. Employing the no-break control character to invoke requests that don't cause breaks is harmless but poor style. See Section 5.9 [Manipulating Filling and Adjustment], page 101.

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 identity of the control character is associated with the environment (see Section 5.31 [Environments], page 204).

`.c2` [*c*] [Request]
Set the no-break control character to *c*. With no argument, the default no-break control character ''' is restored. The identity of the no-break control character is associated with the environment (see Section 5.31 [Environments], page 204).

When writing a macro, you might wish to know which control character was used to call it.

`\n[.br]` [Register]
This read-only register interpolates 1 if the currently executing macro was called using the normal control character and 0 otherwise. If a macro is interpolated as a string, the `.br` register's value is inherited from the context of the string interpolation. See Section 5.22 [Strings], page 157.

Use this register to reliably intercept requests that imply breaks.

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

Testing the `.br` register outside of a macro definition makes no sense.

### 5.6.2 Invoking Requests

A control character is optionally followed by tabs and/or spaces and then an identifier naming a request or macro. The invocation of an unrecognized request is interpreted as a macro call. Defining a macro with the same name as a request replaces the request. Deleting a request name with the `rm` request makes it unavailable. The `als` request can alias requests, permitting them to be wrapped or non-destructively replaced. See Section 5.22 [Strings], page 157.

There is no general limit on argument length or quantity. Most requests take one or more arguments, and ignore any they do not expect. A request may be separated from its arguments by tabs or spaces, but only spaces can separate an argument from its successor. Only one between arguments is necessary; any excess is ignored. GNU `troff` does not allow tabs for argument separation.[23]

[23] In compatibility mode, a space is not necessary after a request or macro name of two characters' length. Also, Plan 9 `troff` allows tabs to separate arguments.

Generally, a space *within* a request argument is not relevant, not meaningful, or is supported by bespoke provisions, as with the `tl` request's delimiters (see Section 5.17 [Page Layout], page 128). Some requests, like `ds`, interpret the remainder of the control line as a single argument. See Section 5.22 [Strings], page 157.

Spaces and tabs immediately after a control character are ignored. Commonly, authors structure the source of documents or macro files with them.

```
.de center
.  if \\n[.br] \
.    br
.  ce \\$@
..
.
.
.de right-align
.→if \\n[.br] \
.→→br
.→rj \\$@
..
```

If you assign an empty blank line trap, you can separate macro definitions (or any input lines) with blank lines.

```
.de do-nothing
..
.blm do-nothing  \" activate blank line trap

.de center
.  if \\n[.br] \
.    br
.  ce \\$@
..


.de right-align
.→if \\n[.br] \
.→→br
.→rj \\$@
..

.blm             \" deactivate blank line trap
```

See Section 5.28.3 [Blank Line Traps], page 195.

### 5.6.3 Calling Macros

If a macro of the desired name does not exist when called, it is created, assigned an empty definition, and a warning in category '`mac`' is emitted. Calling an undefined macro *does* end a macro definition naming it as its end macro (see Section 5.24 [Writing Macros], page 168).

To embed spaces *within* a macro argument, enclose the argument in neutral double quotes `"`. Horizontal motion escape sequences are sometimes a better choice for arguments to be formatted as text.

Consider calls to a hypothetical section heading macro '`uh`'.

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

The first line calls `uh` 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, can be found in documents prepared for AT&T `troff`. It can cause surprise when text is adjusted, because `\SP` inserts a *fixed-width*, non-breaking space. GNU `troff`'s `\~` escape sequence inserts an adjustable, non-breaking space.[24]

The foregoing raises the question of how to embed neutral double quotes or backslashes in macro arguments when *those* characters are desired as literals. In GNU `troff`, the special character escape sequence `\[rs]` produces a backslash and `\[dq]` a neutral double quote.

In GNU `troff`'s AT&T compatibility mode, these characters remain available as `\(rs` and `\(dq`, respectively. AT&T `troff` did not consistently define these special characters, but its descendants can be made to support them. See Section 6.2 [Device and Font Description Files], page 244.

If even that is not feasible, options remain. To obtain a literal escape character in a macro argument, you can simply type it if you change or disable the escape character first. See Section 5.6.4 [Using Escape Sequences], page 89. Otherwise, you must escape the escape character repeatedly to a context-dependent extent. See Section 5.24.2 [Copy Mode], page 174.

For the (neutral) double quote, you have recourse to an obscure syntactical feature of AT&T `troff`. Because a double quote can begin a macro argument, the formatter keeps track of whether the current argument was started thus, and doesn't require a space after the double quote that ends it.[25] In the argument list to a macro, a double quote that *isn't* preceded by a space *doesn't* start a macro argument. If not preceded by a double quote that began an argument, this double quote becomes part of the argument.

---

[24] `\~` is fairly portable; see Section 5.38.3 [Other Differences], page 227.

[25] Strictly, you can neglect to close the last quoted macro argument, relying on the end of the control line to do so. We consider this lethargic practice poor style.

Furthermore, within a quoted argument, a pair of adjacent double quotes becomes a literal double quote.

```
.de eq
.  tm arg1:\\$1 arg2:\\$2 arg3:\\$3
.  tm arg4:\\$4 arg5:\\$5 arg6:\\$6
.. \" 4 backslashes on the next line
.eq a" "b c" "de"f\\\\g" h""i "j""k"
     error  arg1:a" arg2:b c arg3:de
     error  arg4:f\g" arg5:h""i arg6:j"k
```

Apart from the complexity of the rules, this traditional solution has the disadvantage that double quotes don't survive repeated argument expansion in AT&T troff or GNU troff's compatibility mode. This can frustrate efforts to pass such arguments intact through multiple macro calls.

```
.cp 1
.de eq
.  tm arg1:\\$1 arg2:\\$2 arg3:\\$3
.  tm arg4:\\$4 arg5:\\$5 arg6:\\$6
..
.de xe
.  eq \\$1 \\$2 \\$3 \\$4 \\$5 \\$6
.. \" 8 backslashes on the next line
.xe a" "b c" "de"f\\\\\\\\g" h""i "j""k"
     error  arg1:a" arg2:b arg3:c
     error  arg4:de arg5:f\g" arg6:h""i
```

Outside of compatibility mode, GNU troff doesn't exhibit this problem because it tracks the nesting depth of interpolations. See Section 5.38 [Implementation Differences], page 224.

### 5.6.4 Using Escape Sequences

Whereas requests must occur on control lines, escape sequences can occur intermixed with text and may appear in arguments to requests, macros, and other escape sequences. An escape sequence is introduced by the escape character, a backslash \ (but see the ec request below). The next character selects the escape's function.

Escape sequences vary in length. Some take an argument, and of those, some have different syntactical forms for a one-character, two-character, or arbitrary-length argument. Others accept *only* an arbitrary-length argument. In the former scheme, a one-character argument follows the function character immediately, an opening parenthesis '(' introduces a two-character argument (no closing parenthesis is used), and an argument of arbitrary length is enclosed in brackets '[]'. In the latter scheme, the user selects a delimiter character. A few escape sequences are idiosyncratic, and support both of the foregoing conventions (\s), designate their own termination sequence (\?), consume input until the next newline (\!, \", \#), or support an additional modifier character (\s again, and \n). As with requests, use of

some escape sequences in source documents may interact poorly with a macro package you use; consult its documentation to learn of "safe" sequences or alternative facilities it provides to achieve the desired result.

If an escape character is followed by a character that does not identify a defined operation, the escape character is ignored (producing a diagnostic of the '`escape`' warning category, which is not enabled by default) and the following character is processed normally.

```
$ groff -Tps -ww
.nr N 12
.ds co white
.ds animal elephant
I have \fI\nN \*(co \*[animal]s,\f[]
said \P.\&\~Pseudo Pachyderm.
     error  warning: escape character ignored before 'P'
    ⇒ I have 12 white elephants, said P. Pseudo Pachyderm.
```

Escape sequence interpolation is of higher precedence than escape sequence argument interpretation. This rule affords flexibility in using escape sequences to construct parameters to other escape sequences.

```
.ds family C\" Courier
.ds style I\" oblique
Choice a typeface \f(\*[family]\*[style]wisely.
    ⇒ Choose a typeface wisely.
```

In the above, the syntax form '`\f(`' accepts only two characters for an argument; the example works because the subsequent escape sequences are interpolated before the selection escape sequence argument is processed, and strings `family` and `style` interpolate one character each.[26]

The escape character is nearly always interpreted when encountered; it is therefore desirable to have a way to interpolate it.

`\e` [Escape sequence]

Interpolate the escape character.

To format a backslash glyph on the output, use the `\[rs]` special character escape sequence. In macro and string definitions, two further input sequences `\\` and `\E` come into play, permitting deferred interpretation of escape sequences. See Section 5.24.2 [Copy Mode], page 174.

Outside of copy mode, escape sequence interpretation can be switched off and back on. This procedure can obviate the need to double the escape character inside macro definitions. See Section 5.24 [Writing Macros], page 168. (This approach is not available if your macro needs to interpolate values at the time it is *defined*—but many do not.)

[26] The omission of spaces before the comment escape sequences is necessary; see Section 5.22 [Strings], page 157.

`.eo` [Request]

Disable the escape mechanism except in copy mode. Once this request is invoked, no input character is recognized as starting an escape sequence in interpretation mode.

`.ec` [*c*] [Request]

Recognize *c* as the escape character. If *c* is absent or invalid, the default escape character '\' is selected.

Changing the escape character globally likely breaks macro packages, since GNU `troff` has no mechanism to "intern" macros, that is, to convert a macro definition into an internal form independent of its representation.[27] When a macro is called, its contents are interpreted literally.

```
.\" This is a simplified version of the `BR` macro from
.\" the man(7) 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]
.  rm result
.  ft R
..
.ec
```

`.ecs` [Request]

`.ecr` [Request]

The `ecs` request stores the escape character for recall with `ecr`. `ecr` sets the escape character to '\' if none has been saved.

Use these requests together to temporarily change the escape character.

### 5.6.5 Delimiters

Some escape sequences that require parameters use delimiters. The neutral apostrophe `'` is a popular choice and shown in this document. The neutral double quote `"` is also commonly seen. Letters, numerals, and leaders can be used. Punctuation characters are likely better choices, except for those defined as infix operators in numeric expressions; see below.

```
\l'1.5i\[bu]' \" draw 1.5 inches of bullet glyphs
```

The following escape sequences don't take arguments and thus are allowed as delimiters: `\SP`, `\%`, `\|`, `\^`, `\{`, `\}`, `\'`, `\``, `\-`, `_`, `\!`, `\?`, `\)`, `\/`, `\,`, `\&`,

[27] TEX does have such a mechanism.

\:, \~, \0, \a, \c, \d, \e, \E, \p, \r, \t, and \u. However, using them this way is discouraged; they can make the input confusing to read.

A few escape sequences, \A, \b, \o, \w, \X, and \Z, accept a newline as a delimiter. Newlines that serve as delimiters continue to be recognized as input line terminators.

```
A caf\o
e\(aa
in Paris
    ⇒ A café in Paris
```

Use of newlines as delimiters in escape sequences is also discouraged.

Finally, the escape sequences \D, \h, \H, \l, \L, \N, \R, \s, \S, \v, and \x prohibit many delimiters.

- the numerals 0-9 and the decimal point .
- the (single-character) operators '+-/*%<>=&:()'
- the space and tab characters
- any escape sequences other than \%, \:, \{, \}, \', \`, \-, _, \!, \/, \c, \e, and \p

Delimiter syntax is complex and flexible primarily for historical reasons; the foregoing restrictions need be kept in mind mainly when using groff in AT&T compatibility mode. GNU troff keeps track of the nesting depth of escape sequence interpolations, so the only characters you need to avoid using as delimiters are those that appear in the arguments you input, not any that result from interpolation. Typically, ' works fine. See Section 5.38 [Implementation Differences], page 224.

```
$ groff -Tps
.de Mw
.  nr wd \w'\\$1'
.  tm "\\$1" is \\n(wd units wide.
..
.Mw Wet'suwet'en
.Mw Wet+200i
.cp 1 \" turn on compatibility mode
.Mw Wet'suwet'en
.Mw Wet'
.Mw Wet+200i
     error  "Wet'suwet'en" is 54740 units wide.
     error  "Wet'+200i" is 42610 units wide.
     error  "Wet'suwet'en" is 15860 units wide.
     error  "Wet'" is 15860 units wide.
     error  "Wet'+200i" is 14415860 units wide.
```

We see here that in compatibility mode, the part of the argument after the ' delimiter escapes from its context and, if nefariously crafted, influences the computation of the *wd* register's value in a surprising way.

## 5.7 Comments

One of the most common forms of escape sequence is the comment.[28]

`\"` [Escape sequence]

Start a comment. Everything up to the next newline is ignored.

This may sound simple, but it can be tricky to keep the comments from interfering with the appearance of the output. If the escape sequence is to the right of some text or a request, that portion of the line is ignored, but spaces preceding it are processed normally by GNU `troff`. This affects only the `ds` and `as` requests and their variants.

One possibly irritating idiosyncrasy is that tabs should not be used to vertically align comments in the source document. Tab characters are not treated as separators between a request name and its first argument, nor between 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
    ⇒ Test
    ⇒
    ⇒ Test
```

To avoid this, it is common to combine the empty request with the comment escape sequence as '`.\"`', causing the input line to be ignored.

Another commenting scheme sometimes seen is three consecutive single quotes (`'''`) at the beginning of a line. This works, but GNU `troff` emits a warning diagnostic (if enabled) about an undefined macro (namely '`''`').

`\#` [Escape sequence]

Start a comment; everything up to and including the next newline is ignored. This `groff` extension was introduced to avoid the problems described above.

```
Test
\# comment
Test
    ⇒ Test Test
```

`.ig` [*end*] [Request]

Ignore input until, in the current conditional block (if any),[29] the macro *end* is called at the start of a control line, or the control line '`..`' is

[28] This claim may be more aspirational than descriptive.

[29] See Section 5.23.4 [Conditional Blocks], page 165.

encountered if *end* is not specified. `ig` is parsed as if it were a macro definition, but its contents are discarded, not stored.[30]

```
hand\c
.de TX
fasting
..
.ig TX
This is part of a large block of input that has been
temporarily(?) commented out.
We can restore it simply by removing the .ig request and
the call of its end macro.
.TX
```

⇒ `handfasting`

## 5.8 Registers

In the `roff` language, numbers can be stored in *registers*. Many built-in registers exist, supplying anything from the date to details of formatting parameters. You can also define your own. See Section 5.5 [Identifiers], page 83, for information on constructing a valid name for a register.

### 5.8.1 Setting Registers

Define registers and update their values with the `nr` request or the `\R` escape sequence.

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

Set register *ident* to *value*. If *ident* doesn't exist, GNU `troff` creates it. In the `\R` escape sequence, the delimiter need not be a neutral apostrophe; see Section 5.6.5 [Delimiters], page 91. It also does not produce an input token in GNU `troff`. See Section 5.36 [Gtroff Internals], page 217.

```
.nr a (((17 + (3 * 4))) % 4)
\n[a]
.\R'a (((17 + (3 * 4))) % 4)'
\n[a]
```

⇒ `1 1`

(Later, we will discuss additional forms of `nr` and `\R` that can change a register's value after it is dereferenced but before it is interpolated. See Section 5.8.3 [Auto-increment], page 97.)

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

[30] Exception: auto-incrementing registers defined outside the ignored region *will* be modified if interpolated with `\n±` inside it. See Section 5.8.3 [Auto-increment], page 97.

```
.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 sequence 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` 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.

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

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

Increment (decrement) register *ident* by *value*. In the `\R` escape sequence, the delimiter need not be a neutral apostrophe; see Section 5.6.5 [Delimiters], page 91.

```
.nr a 1
.nr a +1
\na
    ⇒ 2
```

A leading minus sign in *value* is always interpreted as a decrementation operator, not an algebraic sign. To assign a register a negative value or the negated value of another register, you can force GNU `troff` to interpret '`-`' as a negation or minus, rather than decrementation, operator: enclose it with its operand in parentheses or subtract it from zero.

```
.nr a 7
.nr b 3
.nr a -\nb
\na
    ⇒ 4
.nr a (-\nb)
\na
    ⇒ -3
.nr a 0-\nb
\na
    ⇒ -3
```

If a register's prior value does not exist (the register was undefined), an increment or decrement is applied as if to 0.

`.rr` *ident* [Request]
Remove 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 register *ident1* to *ident2*. If *ident1* doesn't exist, the request is ignored.

`.aln` *new old* [Request]
Create an alias *new* for an existing register *old*, causing the names to refer to the same stored object. If *old* is undefined, a warning in category '`reg`' is produced and the request is ignored. See Section 5.37.1 [Warnings], page 222, for information about the enablement and suppression of warnings.

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

### 5.8.2 Interpolating Registers

Register contents are interpolated with the `\n` escape sequence.

`\n`*i* [Escape sequence]
`\n(`*id* [Escape sequence]
`\n[`*ident*`]` [Escape sequence]
Interpolate register with name *ident* (one-character name *i*, two-character name *id*). `\n` is interpreted even in copy mode (see Section 5.24.2 [Copy Mode], page 174). If the register is undefined, it is created, assigned a value of '`0`', that value is interpolated, and a warning in category '`reg`' is emitted. See Section 5.37.1 [Warnings], page 222, for information about the enablement and suppression of warnings.

```
.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
```

### 5.8.3 Auto-increment

Registers can also be incremented or decremented by a configured amount at the time they are interpolated. The value of the increment is specified with a third argument to the `nr` request, and a special interpolation syntax is used to alter and then retrieve the register's value. Together, these features are called *auto-increment*.[31]

`.nr` *ident value incr* [Request]
Set register *ident* to *value* and its auto-incrementation amount to to *incr*. The `\R` escape sequence doesn't support an *incr* argument.

Auto-incrementation is not *completely* automatic; the `\n` escape sequence in its basic form never alters the value of a register. To apply auto-incrementation to a register, interpolate it with '`\n`±'.

`\n+`*i* [Escape sequence]
`\n-`*i* [Escape sequence]
`\n+(`*id* [Escape sequence]
`\n-(`*id* [Escape sequence]
`\n+[`*ident*`]` [Escape sequence]
`\n-[`*ident*`]` [Escape sequence]
Increment or decrement *ident* (one-character name *i*, two-character name *id*) by the register's auto-incrementation value and then interpolate the new register value. If *ident* has no auto-incrementation value, interpolate as with `\n`.

[31] A negative auto-increment can be considered an "auto-decrement".

```
.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]
    ⇒ 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, assign the register's value to itself by interpolating it, and specify the desired increment normally. Apply an increment of '`0`' to disable auto-incrementation of the register.

### 5.8.4 Assigning Register Formats

A writable register's value can be interpolated in several number formats. By default, conventional Arabic numerals are used. Other formats see use in sectioning and outlining schemes and alternative page numbering arrangements.

`.af` *reg fmt* [Request]

Use number format *fmt* when interpolating register *reg*. Valid number formats are as follows.

`0...` Arabic numerals 0, 1, 2, and so on. Any decimal digit is equivalent to '`0`'; the formatter merely counts the digits specified. Multiple Arabic numerals in *fmt* cause interpolations to be zero-padded on the left if necessary to at least as many digits as specified (interpolations never truncate a register value). A register with format '`00`' interpolates values 1, 2, 3 as '`01`', '`02`', '`03`'. The default format for all writable registers is '`0`'.

`I` Uppercase Roman numerals: 0, I, II, III, IV, ...

`i` Lowercase Roman numerals: 0, i, ii, iii, iv, ...

`A` Uppercase letters: 0, A, B, C, ..., Z, AA, AB, ...

`a` Lowercase letters: 0, a, b, c, ..., z, aa, ab, ...

Omitting *fmt* causes a warning in category '`missing`'. See Section 5.37.1 [Warnings], page 222, for information about the enablement and suppression of warnings. Specifying an unrecognized format is an error.

Zero values are interpolated as '`0`' in non-Arabic formats. Negative quantities are prefixed with '`-`' irrespective of format. In Arabic formats, the sign supplements the field width. If *reg* doesn't exist, it is created with a zero value.

```
.nr a 10
.af a 0           \" the default format
\na,
.af a I
\na,
.af a 321
.nr a (-\na)
\na,
.af a a
\na
    ⇒ 10, X, -010, -j
```

The representable extrema in the 'i' and 'I' formats correspond to Arabic ±39,999. GNU troff uses 'w' and 'z' to represent 5,000 and 10,000 in Roman numerals, respectively, following the convention of AT&T troff—currently, the correct glyphs for Roman numerals five thousand (U+2181) and ten thousand (U+2182) are not used.

Assigning the format of a read-only register is an error. Instead, copy the read-only register's value to, and assign the format of, a writable register.

\g*r* [Escape sequence]
\g(*rg* [Escape sequence]
\g[*reg*] [Escape sequence]

Interpolate the format of the register *reg* (one-character name *r*, two-character name *rg*). Zeroes represent Arabic formats. If *reg* is not defined, *reg* is not created and nothing is interpolated. \g is interpreted even in copy mode (see Section 5.24.2 [Copy Mode], page 174).

GNU troff interprets only Arabic numerals. The Roman numeral or alphabetic formats cannot be used as operands to arithmetic operators in expressions (see Section 5.4 [Numeric Expressions], page 79). For instance, it may be desirable to test the page number independently of its format.

```
.af % i \" front matter
.de header-trap
.  \" To test the page number, we need it in Arabic.
.  ds saved-page-number-format \\g%\"
.  af % 0
.  nr page-number-in-decimal \\n%
.  af % \\*[saved-page-number-format]
.  ie \\n[page-number-in-decimal]=1 .do-first-page-stuff
.  el \{\
.    ie o .do-odd-numbered-page-stuff
.    el   .do-even-numbered-page-stuff
.  \}
.  rm saved-page-number-format
..
.wh 0 header-trap
```

### 5.8.5 Built-in Registers

Predefined registers whose identifiers start with a dot are read-only. Many are Boolean-valued, interpolating a true or false value testable with the `if`, `ie`, or `while` requests. Some read-only registers are string-valued, meaning that they interpolate text.

A register name (without the dot) is often associated with a request of the same name. A complete listing of all built-in registers can be found in Appendix E [Register Index], page 269.

We present here a few built-in registers that are not described elsewhere in this manual; they have to do with invariant properties of GNU `troff`, or obtain information about the formatter's command-line options or the operating environment. Date- and time-related registers are set per the local time as determined by *localtime*(3) when the formatter launches. This initialization can be overridden by `SOURCE_DATE_EPOCH` and `TZ`; see Section 2.2 [Environment], page 12.

`\n[.F]` Name of input file (string-valued).

`\n[.R]` Count of available unused registers; always 10,000 in GNU `troff`.[32]

`\n[.U]` Unsafe mode enablement (Boolean-valued); see `groff -U` option (Section 2.1 [Groff Options], page 7).

`\n[seconds]`
Count of seconds elapsed in the minute (0–60).

`\n[minutes]`
Count of minutes elapsed in the hour (0–59).

`\n[hours]`
Count of hours elapsed since midnight (0–23).

`\n[dw]` Day of the week (1–7; 1 is Sunday).

`\n[dy]` Day of the month (1–31).

`\n[mo]` Month of the year (1–12).

`\n[year]` Gregorian year.

`\n[yr]` Gregorian year minus 1900. This register is incorrectly documented in the AT&T `troff` manual as storing the last two digits of the current year. That claim stopped being true in 2000. Old `troff` input that looks like:

```
'\" The year number is a surprise after 1999.
This document was formatted in 19\n(yr.
```

can be corrected to:

```
This document was formatted in \n[year].
```

[32] GNU `troff` dynamically allocates memory for as many registers as required.

or, for portability across many roff programs, to the following.

```
.nr y4 1900+\n(yr
This document was formatted in \n(y4.
```

\n[.c]
\n[c.] Input line number. 'c.' is a writable synonym, affecting subsequent interpolations of both '.c' and 'c.'.

\n[.x] Major version number of the running GNU troff formatter. For example, if the version number is 1.23.0, then .x contains '1'.

\n[.y] Minor version number of the running GNU troff formatter. For example, if the version number is 1.23.0, then .y contains '23'.

\n[.Y] Revision number of the running GNU troff formatter. For example, if the version number is 1.23.0, then .Y contains '0'.

\n[$$] Process identifier (PID) of the GNU troff program in its operating environment.

\n[.g] Always true in GNU troff (Boolean-valued). Documents can use this to ask the formatter if it claims groff compatibility.

\n[.A] Approximate output is being formatted (Boolean-valued); see groff -a option (Section 2.1 [Groff Options], page 7).

\n[.P] Output page selection status (Boolean-valued); see groff -o option (Section 2.1 [Groff Options], page 7).

\n[.T] Indicator of output device selection (Boolean-valued); see groff -T option (Section 2.1 [Groff Options], page 7).

## 5.9 Manipulating Filling and Adjustment

When an output line is pending (see below), a break moves the drawing position to the beginning of the next text baseline, interrupting filling. Various ways of causing breaks were shown in Section 5.1.4 [Breaking], page 68. The br request likewise causes a break. Several other requests imply breaks: bp, ce, cf, fi, fl, in, nf, rj, sp, ti, and trf. If the no-break control character is used with any of these requests, GNU troff suppresses the break; instead the requested operation takes effect at the next break. ''br' does nothing.

```
.ll 55n
This line is normally filled and adjusted.
.br
A line's alignment is decided
'ce \" Center the next input line (no break).
when it is output.
This line returns to normal filling and adjustment.
    ⇒ This line is normally filled and adjusted.
    ⇒    A line's alignment is decided when it is output.
    ⇒ This line returns to normal filling and adjustment.
```

Output line properties like page offset, indentation, and adjustment are not determined until the line has been broken. An output line is said to be *pending* if some input has been collected but an output line corresponding to it has not yet been written; such an output line is also termed *partially collected*. If no output line is pending, it is as if a break has already happened; additional breaks, whether explicit or implicit, have no effect. If the vertical drawing position is negative—as it is when the formatter starts up—a break starts a new page (even if no output line is pending) unless an end-of-input macro is being interpreted. See Section 5.28.5 [End-of-input Traps], page 195.

`.br` [Request]

Break the line: emit any pending output line without adjustment.

```
foo bar
.br
baz
'br
qux
    ⇒ foo bar
    ⇒ baz qux
```

Sometimes you want to prevent a break within a phrase or between a quantity and its units.

`\~` [Escape sequence]

Insert an unbreakable space that is adjustable like an ordinary space. It is discarded from the end of an output line if a break is forced.

```
Set the output speed to\~1.
There are 1,024\~bytes in 1\~KiB.
J.\~F.\~Ossanna wrote the original CSTR\~#54.
```

By default, GNU `troff` fills text and adjusts it to both margins. Filling can be disabled via the `nf` request and re-enabled with the `fi` request.

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

Enable filling of output lines; a pending output line is broken. The read-only register `.u` is set to 1. The filling enablement status, sometimes called *fill mode*, is associated with the environment (see Section 5.31 [Environments], page 204). See Section 5.16 [Line Continuation], page 127, for interaction with the `\c` escape sequence.

`.nf` [Request]

Disable filling of output lines: the output line length (see Section 5.15 [Line Layout], page 124) is ignored and output lines are broken where the input lines are. A pending output line is broken and adjustment is suppressed. The read-only register `.u` is set to 0. The filling enablement status is associated with the environment (see Section 5.31 [Environments],

page 204). See Section 5.16 [Line Continuation], page 127, for interaction with the \c escape sequence.

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

Enable output line adjustment in *mode*, taking effect when the pending (or next) output line is broken. Adjustment is suppressed when filling is. *mode* can have one of the following values.

b
n
: Adjust "normally": to both margins. This is the GNU troff default.

c
: Center filled text. Contrast with the ce request, which centers text without filling it.

l
: Align text to the left margin, producing what is sometimes called ragged-right text.

r
: Align text to the right margin, producing ragged-left text.

*mode* can also be a value previously stored in the .j register. Using ad without an argument is the same as '.ad \n[.j]'; unless filling is disabled, GNU troff resumes adjusting lines in the same way it did before adjustment was disabled by invocation of the na request.

```
.ll 48n
.de AD
.  br
.  ad \\$1
..
.
.de NA
.  br
.  na
..
.
left
.AD r
.nr ad \n(.j
right
.AD c
center
.NA
left
.AD
center
.AD \n(ad
right
```

```
⇒ left
⇒                                                    right
⇒                          center
⇒ left
⇒                          center
⇒                                                    right
```

The adjustment mode and enablement status are encoded in the read-only register `.j`. These parameters are associated with the environment (see Section 5.31 [Environments], page 204).

The value of `.j` for any adjustment mode is an implementation detail and should not be relied upon as a programmer's interface. Do not write logic to interpret or perform arithmetic on it.

`.na` [Request]

Disable output line adjustment. This produces the same output as alignment to the left margin, but the value of the adjustment mode register `.j` is altered differently. The adjustment mode and enablement status are associated with the environment (see Section 5.31 [Environments], page 204).

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

Break, adjusting the line per the current adjustment mode. `\p` schedules a break with adjustment at the next word boundary. The escape sequence is itself neither a break nor a space of any kind; it can thus be placed in the middle of a word to cause a break at the end of that word.

Breaking with immediate adjustment can produce ugly results since GNU `troff` doesn't have a sophisticated paragraph-building algorithm, as TEX has, for example. Instead, GNU `troff` fills and adjusts a paragraph line by line.

```
.ll 4.5i
This is an uninteresting sentence.
This is an uninteresting sentence.\p
This is an uninteresting sentence.
```

is formatted as follows.

```
This  is  an uninteresting sentence.  This is
an          uninteresting          sentence.
This is an uninteresting sentence.
```

To clearly present the next couple of requests, we must introduce the concept of "productive" input lines. A *productive input line* is one that directly produces formatted output. Text lines produce output, as do control lines containing requests like `tl` or escape sequences like `\D`. Macro calls are not *directly* productive, and thus not counted, but their interpolated contents can be. Empty requests, and requests and escape sequences that define registers or strings or alter the formatting environment (as with changes to

the size, face, height, slant, or color of the type) are not productive. The output line continuation escape sequence \c "connects" two input lines that would otherwise be counted separately. See Section 5.16 [Line Continuation], page 127.

```
.de hello
Hello, world!
..
.ce \" center output of next productive input line
.
.nr junk-reg 1
.ft I
Chorus: \c
.ft
.hello
Went the day well?
  ⇒                   Chorus: Hello, world!
  ⇒ Went the day well?
```

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

Break (unless the no-break control character is used), center the output of the next *nnn* productive input lines without filling, then break again (regardless of the control character). The count of lines remaining to be centered is stored in the read-only register `.ce` and is associated with the environment (see Section 5.31 [Environments], page 204).

While the '`.ad c`' request also centers text, it fills the text as well. The following example demonstrates the difference.

```
.de FR
This is a small text fragment that shows the differences
between the `.ce' and the `.ad c' requests.
..
.ll 4i
.ce 1000
.FR
.ce 0

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

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 basis 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 illustrates a common idiom of turning centering on for a quantity of lines far in excess of what is required, and off again after the text to be centered. This technique relieves humans of counting lines for requests that take a count of input lines as an argument.

`.rj` [*nnn*] [Request]
`\n[.rj]` [Register]

Break (unless the no-break control character is used), align the output of the next *nnn* productive input lines to the right margin without filling, then break again (regardless of the control character). The count of lines remaining to be centered is stored in the read-only register `.rj` and is associated with the environment (see Section 5.31 [Environments], page 204).

`.ss` *word-space-size* [*additional-sentence-space-size*] [Request]
`\n[.ss]` [Register]
`\n[.sss]` [Register]

Set the sizes of spaces between words and sentences.[33] Their units are twelfths of the space width of the current font. Initially both the *word-space-size* and *additional-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 an output line undergoes adjustment, such spaces may increase in width.

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

The read-only registers `.ss` and `.sss` hold the values of minimal inter-word space and additional inter-sentence space, respectively. These parameters are associated with the environment (see Section 5.31 [Environments], page 204), and rounded down to the nearest multiple of 12 on terminal output devices.

Additional inter-sentence space is used 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

[33] See Section 5.1.1 [Filling], page 65, and Section 5.1.2 [Sentences], page 66, for the definitions of word and sentence boundaries, respectively.

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.

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.

```
.ll 48n
1.\~J. Fict. Ch. Soc. 6 (2020), 3\[en]14.
.ss 12 48 \" applies to next sentence ending
Reprints no longer available through FCS.
.ss 12 \" go back to normal
2.\~Better known for other work.
    ⇒ 1.  J.  Fict. Ch. Soc. 6 (2020), 3-14.  Reprints
    ⇒ no longer available through FCS.      2.  Better
    ⇒ known for other work.
```

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

## 5.10 Manipulating Hyphenation

When filling, GNU `troff` hyphenates words as needed at user-specified and automatically determined hyphenation points. The machine-driven determination of hyphenation points in words requires algorithms and data, and is susceptible to conventions and preferences. Before tackling such *automatic hyphenation*, let us consider how hyphenation points can be set manually.

Explicitly hyphenated words such as "mother-in-law" are always eligible for breaking after each of their hyphens. Relatively few words in a language offer such obvious break points, however, and automatic hyphenation is not perfect, particularly for unusual words found in technical literature. We may wish to instruct GNU `troff` how to hyphenate specific 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 within-word placement restrictions imposed by the `hy` request (see below).

Hyphenation exceptions specified with the `hw` request are associated with the hyphenation language (see the `hla` request below) and environment

(see Section 5.31 [Environments], page 204); invoking 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 breakable in sensible places without hyphens.

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

To tell GNU `troff` how to hyphenate words as they occur in input, use the `\%` escape sequence; it is the default *hyphenation character*. Each instance within a word indicates to GNU `troff` that the word may be hyphenated at that point, while prefixing a word with this escape sequence prevents it from being otherwise hyphenated. This mechanism affects only that occurrence of the word; to change the hyphenation of a word for the remainder of input processing, use the `hw` request.

GNU `troff` regards the escape sequences `\X` and `\Y` as starting a word; that is, the `\%` escape sequence 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 5.34 [Postprocessor Access], page 212.

`\:` inserts a non-printing break point; that is, a word can break there, but the soft hyphen glyph (see below) is not written to the output if it does. This escape sequence is an input word boundary, so the remainder of the word is subject to hyphenation as normal.

You can use `\:` and `\%` in combination to control breaking of a file name or URL or to permit hyphenation only after certain explicit hyphens within a word.

```
The \%Lethbridge-Stewart-\:\%Sackville-Baggins divorce
was, in retrospect, inevitable once the contents of
\%/var/log/\:\%httpd/\:\%access_log on the family web
server came to light, revealing visitors from Hogwarts.
```

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

Change the hyphenation character to *char*. This character then works as the `\%` escape sequence normally does, and thus no longer appears in the output.[34] Without an argument, `hc` resets the hyphenation character to `\%` (the default). The hyphenation character is associated with the environment (see Section 5.31 [Environments], page 204).

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

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

Set the *soft hyphen character*,[35] inserted when a word is hyphenated automatically or at a hyphenation character, to *glyph*.[36] If the argument is omitted, the soft hyphen glyph is set to the default, `\[hy]`. If the selected glyph does not exist in the font in use at a potential hyphenation point, then the line is not broken at that point. Neither character definitions (specified with the `char` and similar requests) nor translations (specified with the `tr` request) are considered when assigning the soft hyphen glyph.

Several requests influence automatic hyphenation. Because conventions vary, a variety of hyphenation modes is available to the `hy` request; these determine whether hyphenation will apply to a word prior to breaking a line at the end of a page (more or less; see below for details), and at which positions within that word automatically determined hyphenation points are permissible. The 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 due to a limit on consecutive hyphenated lines (`hlm`), a minimum line length threshold, certain minimum length (`hym`), or because the line can instead be adjusted with additional inter-word space (`hys`).

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

Set automatic hyphenation mode to *mode*, an integer encoding conditions for hyphenation; if omitted, '`1`' is implied. The hyphenation mode is available in the read-only register '`.hy`'; it is associated with the environment (see Section 5.31 [Environments], page 204). The default hyphenation mode depends on the localization package loaded when GNU `troff` starts up; see the `hpf` request below.

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, favoring compatibility with older implementations over 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.[37] The entries in the following table are termed *values*; the sum of the desired values is the *mode*.

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

[36] "Soft" because it appears in output only where a hyphenation break is performed; a "hard" hyphen, as in "long-term", always appears.

[37] 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.

`0` disables hyphenation.

`1` enables hyphenation except after the first and before the last character of a word.

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

`2` disables hyphenation of the last word on a page,[38] even for manually hyphenated words.

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

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

`16` enables hyphenation before the last character of a word.

`32` enables hyphenation after the first character of a word.

Apart from value 2, restrictions imposed by the hyphenation mode are *not* respected for words whose hyphenations have been specified with the hyphenation character ('`\%`' by default) or the `hw` request.

Nonzero values in the previous table are additive. For example, mode 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 nonzero even mode.

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
```

[38] Hyphenation is prevented if the next page location trap is closer to the vertical drawing position than the next text baseline would be. GNU `troff` automatically inserts an implicit vertical position trap at the end of each page to cause a page transition. Users or macro packages can set such traps explicitly to prevent hyphenation of the last word in a column in multi-column page layouts or before floating figures or tables. See Section 5.28.1.1 [Page Location Traps], page 188.

instead of the correct 'split- ting'. 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. 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 for more information on GNU troff's language macro files.

| language | pattern name | left min | right min |
|---|---|---|---|
| Czech | cs | 2 | 2 |
| English | en | 2 | 3 |
| French | fr | 2 | 3 |
| German traditional | det | 2 | 2 |
| German reformed | den | 2 | 2 |
| Italian | it | 2 | 2 |
| Swedish | sv | 1 | 2 |

Hyphenation exceptions within pattern files (i.e., the words within a TEX \hyphenation group) obey the hyphenation restrictions given by hy.

.nh [Request]

Disable automatic 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*, which is sought in the same way that macro files are with the mso request or the -m*name* 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 backward 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 (see below). The `hpf` request is usually invoked by a localization file loaded by the `troffrc` file. By default, `troffrc` loads the localization file for English.[39] For Western languages, the localization file sets the hyphenation mode and loads hyphenation patterns and exceptions.

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 file loaded at startup, 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 an ordinary character (not a special character escape sequence) other than a digit or a space. The request is ignored if given no arguments.

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.

[39] As of `groff` 1.23.0, localization files for Czech (`cs`), German (`de`), English (`en`), French (`fr`), Italian (`it`), Japanese (`ja`), Swedish (`sv`), and Chinese (`zh`) exist.

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

Without these assignments, GNU troff treats the German word 'Kindergärten' (the plural form of 'kindergarten') as two words 'kinderg' and 'rten' because the hyphenation code of the umlaut a is zero by default, just like a space. 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 a localization file, which is turn loaded by the `troffrc` or `troffrc-end` file; see the `hpf` request above.

The hyphenation language is available in the read-only string-valued register '`.hla`'; it is associated with the environment (see Section 5.31 [Environments], page 204).

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

Set the maximum quantity 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 environment (see Section 5.31 [Environments], page 204). Only lines output from a given environment count toward the maximum associated with that environment. Hyphens resulting from `\%` are counted; explicit hyphens are not.

The `.hlm` read-only register stores this maximum. The count of immediately preceding consecutive hyphenated lines is available in the read-only 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 environment (see Section 5.31 [Environments], page 204).

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

The hyphenation margin is available in the `.hym` read-only 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 environment (see Section 5.31 [Environments], page 204).

A negative argument resets the hyphenation space adjustment threshold to zero, emitting a warning in category '`range`'.

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

## 5.11 Manipulating Spacing

`.sp` [*distance*] [Request]
Space downward by *distance*. With no argument, the drawing position is advanced downward by one vee. Inside a diversion, any *distance* argument is ignored. Otherwise, a negative argument moves the drawing position up the page. This request causes a break. The default scaling unit is '`v`'. If *distance* is not specified, '`1v`' is assumed.

You may wish to use the following macros to set the baseline of the next output text at a given distance from the top or the bottom of the page. We subtract one line height (`\n[.v]`) because the `|` operator moves to one vee below the page top (see Section 5.4 [Numeric Expressions], page 79).

```
.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 next text baseline will be at 10 cm from the bottom edge of the paper.

If a vertical position trap is sprung during execution of `sp`, the amount of vertical space after the trap is discarded.

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

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

To protect `sp` against vertical position traps, use the `vpt` request to disable them. See Section 5.28.1 [Vertical Position Traps], page 188.

```
.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 read-only register `.L` contains the current line spacing setting. The line spacing is associated with the environment (see Section 5.31 [Environments], page 204).

See Section 5.20.1 [Changing the Type Size], page 151, for the requests `vs` and `pvs` as alternatives to `ls`.

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

Sometimes, extra vertical spacing is needed only occasionally, for instance to allow room for a tall construct like an inline equation. The `\x` escape sequence takes a delimited measurement (like '`\x'3p'`'); the default scaling unit is 'v'. If the measurement is positive, extra vertical space is inserted below the current line; a negative measurement adds space above. If `\x` is used multiple times on the same output line, the maxima of the positive

and negative adjustments are used. The delimiter need not be a neutral apostrophe; see Section 5.6.5 [Delimiters], page 91.

The `.a` read-only register contains the most recent (non-negative) extra vertical line space.

Use of `\x` can be necessary in combination with the `\b` escape sequence, as the following example shows.

```
This is a test of \[rs]b.
.br
This is a test of \[rs]b.
.br
This is a test of \b'xyz'\x'-1m'\x'1m'.
.br
This is a test of \[rs]b.
.br
This is a test of \[rs]b.
```

produces

```
This is a test of \b.
This is a test of \b.
                  x
This is a test of y.
                  z
This is a test of \b.
This is a test of \b.
```

`.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 5.18 [Page Control], page 129). This mode ends when actual text is output or the `rs` request is encountered, which ends no-space mode. The read-only 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).

## 5.12 Tabs and Fields

A tab character (ISO code point 9, EBCDIC code point 5) causes a horizontal movement to the next tab stop, if any.

`\t` [Escape sequence]

Interpolate a tab in copy mode.

`.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. The default scaling unit is '`m`'. Invoking `ta` without an argument removes all tab stops. GNU `troff`'s startup value is '`T 0.5i`'.

Tab stops can be specified absolutely—as distances from the left margin. The following example sets six tab stops, one every 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
```

GNU `troff` supports an extended syntax to specify repeating tab stops. These stops appear after a '`T`' argument. Their values are always taken as distances relative to the previous tab stop. This is the idiomatic way to specify tab stops at equal intervals in `groff`. The following is, yet again, the same as the previous examples. It does more, in fact, since it defines an infinite number of tab stops at one-inch intervals.

```
.ta T 1i
```

Now we are ready to interpret the full syntax given above. The `ta` request sets tabs at positions *n1*, *n2*, ..., *nn*, then at *nn*+*r1*, *nn*+*r2*, ..., *nn*+*rn*, then at *nn*+*rn*+*r1*, *nn*+*rn*+*r2*, ..., *nn*+*rn*+*rn*, and so on.

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

Text written to a tab column (i.e., between two tab stops, or between a tab stop and an output line boundary) may be aligned to the right or left, or centered in the column. This alignment is determined by appending '`R`', '`L`', or '`C`' to the tab specifier. The default is '`L`'.

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

The beginning of an output line is not a tab stop; the text that begins an output line is placed according to the configured alignment and indentation; see Section 5.9 [Manipulating Filling and Adjustment], page 101, and Section 5.15 [Line Layout], page 124.

A tab stop is converted into a non-breakable horizontal movement that cannot be adjusted.

```
.ll 2i
.ds foo a\tb\tc
.ta T 1i
\*[foo]
     error  warning: cannot break line
     ⇒ a         b         c
```

The above creates a single output line that is a bit longer than two inches (we use a string to show exactly where the tab stops are). Now consider the following.

```
.ll 2i
.ds bar a\tb c\td
.ta T 1i
\*[bar]
     error  warning: cannot adjust line
    ⇒ a         b
    ⇒ c       d
```

GNU `troff` first converts the line's tab stops into unbreakable horizontal movements, then breaks after '`b`'. This usually isn't what you want.

Superfluous tab characters—those that do not correspond to a tab stop—are ignored except for the first, which delimits the characters belonging to the last tab stop for right-alignment or centering.

```
.ds Z   foo\tbar\tbaz
.ds ZZ  foo\tbar\tbazqux
.ds ZZZ foo\tbar\tbaz\tqux
.ta 2i 4iR
\*[Z]
.br
\*[ZZ]
.br
\*[ZZZ]
.br
    ⇒ foo                 bar              baz
    ⇒ foo                 bar           bazqux
    ⇒ foo                 bar              bazqux
```

The first line right-aligns "baz" within the second tab stop. The second line right-aligns "bazqux" within it. The third line right-aligns only "baz" because of the additional tab character, which marks the end of the text occupying the last tab stop defined.

Tab stops are associated with the environment (see Section 5.31 [Environments], page 204).

The read-only register `.tabs` contains a string representation of the current tab settings suitable for use as an argument to the `ta` request.[40]

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

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

Normally, GNU `troff` writes no glyph when moving to a tab stop (some output devices may explicitly output space characters to achieve this mo-

[40] Plan 9 `troff` uses the register `.S` for this purpose.

tion). A *tab repetition character* can be specified with the `tc` request, causing GNU `troff` to write as many instances of *fill-glyph* as are necessary to occupy the interval from the current horizontal location to the next tab stop. With no argument, GNU `troff` reverts to the default behavior. The tab repetition character is associated with the environment (see Section 5.31 [Environments], page 204).[41] Only a single *fill-glyph* is recognized; any excess is ignored.

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

If *n* is missing or nonzero, activate *line-tabs*; deactivate it otherwise (the default). Active line-tabs cause GNU `troff` to compute tab distances relative to the start of the output line instead of the input line.

```
.de Tabs
.  ds x a\t\c
.  ds y b\t\c
.  ds z c
.  ta 1i 3i
\\*x
\\*y
\\*z
..
.Tabs
.br
.linetabs
.Tabs
    ⇒ a          b          c
    ⇒ a          b                     c
```

Line-tabs activation is associated with the environment (see Section 5.31 [Environments], page 204). The read-only register `.linetabs` interpolates 1 if line-tabs are active, and 0 otherwise.

### 5.12.1 Leaders

Sometimes it is desirable to fill a tab stop with a given glyph, but also use tab stops normally on the same output line. An example is a table of contents entry that uses dots to bridge the entry name with its page number, which is itself aligned within a tab stop. The `roff` language provides *leaders* for this purpose.[42]

A leader character (ISO and EBCDIC code point 1, also known as SOH or "start of heading"), behaves similarly to a tab character: it moves to the next tab stop. The difference is that for this movement, the default fill glyph is a period '`.`'.

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

[42] 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.

`\a` [Escape sequence]
Interpolate a leader in copy mode.

`.lc` [*fill-glyph*] [Request]
When writing a leader, GNU `troff` fills the space to the next tab stop with dots '`.`'. A different *leader repetition character* can be specified with the `lc` request, causing GNU `troff` to write as many instances of *fill-glyph* as are necessary to occupy the interval from the current horizontal location to the next tab stop. With no argument, GNU `troff` treats leaders the same as tabs. The leader repetition character is associated with the environment (see Section 5.31 [Environments], page 204).[43] Only a single *fill-glyph* is recognized; any excess is ignored.

A table of contents, for example, may define tab stops after a section number, a title, and a gap to be filled with leader dots. The page number follows the leader, after a right-aligned final tab stop wide enough to house the largest page number occurring in the document.

```
.ds entry1 19.\tThe Prophet\a\t98
.ds entry2 20.\tAll Astir\a\t101
.ta .5i 4.5i +.5iR
.nf
\*[entry1]
\*[entry2]
    ⇒ 19.  The Prophet.............................  98
    ⇒ 20.  All Astir...............................  101
```

### 5.12.2 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 an adjustable space similar to TEX's `\hss` command (thus it can even be negative) to make the sum of all substring lengths plus the adjustable 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 environment (see Section 5.31 [Environments], page 204).

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

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

## 5.13 Character Translations

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 5.36 [Gtroff Internals], page 217, for more on this process).

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

Translate character *a* to glyph *b*, character *c* to glyph *d*, and so on. If there is an odd number of characters in the argument, the last one is translated to a fixed-width space (the same one obtained by the '`\ `' escape).

The `trin` request is identical to `tr`, but when you unformat a diversion with `asciify` it ignores the translation. See Section 5.29 [Diversions], page 197, 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 `\~` escape sequences (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\&`' (an arbitrary character *c* followed by the dummy character) maps this character to nothing.

  ```
  .tr a\&
  foo bar
      ⇒ foo br
  ```

  It is even possible to map the space character to nothing:

```
.tr aa \&
foo bar
    ⇒ foobar
```

As shown in the example, the space character can't be the first character/glyph pair as an argument of `tr`. Additionally, it is not possible to map the space character to any other glyph; requests like '`.tr aa x`' undo '`.tr aa \&`' instead.

If justification is active, lines are justified in spite of the 'empty' space character (but there is no minimal distance, i.e., the space character, between words).

- After an output glyph has been constructed (this happens at the moment immediately before the glyph is appended to an output glyph list, either by direct output, in a macro, diversion, or string), it is no longer affected by `tr`.
- Translating character to glyphs where one of them or both are undefined is possible also; `tr` does not check whether the entities in its argument do exist.

  See Section 5.36 [Gtroff Internals], page 217.
- `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 5.29 [Diversions], page 197.

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'.

## 5.14 troff and nroff Modes

Historically, `nroff` and `troff` were two separate programs; the former for terminal output, the latter for typesetters. GNU `troff` merges both functions into one executable[44] that sends its output to a device driver (`grotty` for terminal devices, `grops` for PostScript, and so on) which interprets this intermediate output format. When discussing AT&T `troff`, it makes sense to talk about `nroff` *mode* and `troff` *mode* since the differences are hard-coded. GNU `troff` takes information from device and font description files without handling requests specially if a terminal output device is used, so such a strong distinction is unnecessary.

Usually, a macro package can be used with all output devices. Nevertheless, it is sometimes necessary to make a distinction between terminal and non-terminal devices: GNU `troff` provides two built-in conditions 'n' and 't' for the `if`, `ie`, and `while` requests to decide whether GNU `troff` 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 GNU `troff` (*not* `groff`) is started with the `-R` switch to avoid loading of the startup files `troffrc` and `troffrc-end`. Without `-R`, GNU `troff` stays in `troff` mode if the output device is not a terminal (e.g., 'ps').

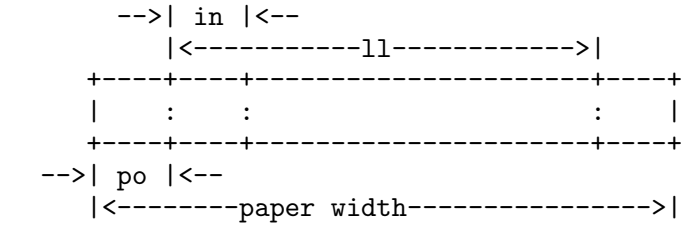
`.nroff` [Request]
Make the 'n' built-in condition true (and the 't' built-in condition false) for `if`, `ie`, and `while` conditional requests. This is the default if GNU `troff` uses a terminal output device; the code for switching to `nroff` mode is in the file `tty.tmac`, which is loaded by the startup file `troffrc`.

See Section 5.23 [Conditionals and Loops], page 161, for more details on built-in conditions.

[44] A GNU `nroff` program is available for convenience; it calls GNU `troff` to perform the formatting.

## 5.15 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.

The right margin is not explicitly configured; the combination of page offset and line length provides the information necessary to derive it.

A simple demonstration:

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

Result:

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

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

Set page offset to *offset* (or increment or decrement its current value by *offset*). If invoked without an argument, the page offset is restored to the value before the previous `po` request. This request does not cause a break; the page offset in effect when an output line is broken prevails (see Section 5.9 [Manipulating Filling and Adjustment], page 101). The initial value is 1 i and the default scaling unit is '`m`'. On terminal devices, the page offset is set to zero by a driver-specific macro file, `tty.tmac`. The current page offset can be found in the read-only register '`.o`'. This request is incorrectly documented in the AT&T `troff` manual as using a default scaling unit of '`v`'.

```
.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`'.

If a negative indentation value is specified (which is not allowed), `gtroff` emits a warning in category '`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 register '`.i`'. The indentation is associated with the environment (see Section 5.31 [Environments], page 204).

.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 environment (see Section 5.31 [Environments], page 204). 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 in category '`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 register `.in` is the indentation that applies to the current output line.

The difference between `.i` and `.in` is that the latter takes into account whether a partially collected line still uses the old indentation value or a temporary indentation value is active.

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

Set the line length to *length* (or increment or decrement the current value by *length*). Initially, the line length is set to 6.5 i. The effect of `ll` is delayed until a partially collected line (if it exists) is output. The default scaling indicator is '`m`'.

If `ll` is called without an argument, the line length is reset to the previous value before the last call to `ll`. If a negative line length is specified (which is not allowed), `gtroff` emits a warning in category '`range`' and sets the line length to zero. The line length is associated with the environment (see Section 5.31 [Environments], page 204).

The current line length (as set by `ll`) can be found in the read-only register '`.l`'. The read-only 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.

## 5.16 Line Continuation

When filling is enabled, input and output line breaks generally do not correspond. The roff language therefore distinguishes input and output line continuation.

\RET [Escape sequence]

\RET (a backslash immediately followed by a newline) suppresses the effects of that newline in the input. The next input line thus retains the classification of its predecessor as a control or text line. \RET is useful for managing line lengths in the input during document maintenance; you can break an input line in the middle of a request invocation, macro call, or escape sequence. Input line continuation is invisible to the formatter, with two exceptions: the | operator recognizes the new input line (see Section 5.4 [Numeric Expressions], page 79), and the input line counter register .c is incremented.

```
.ll 50n
.de I
.  ft I
.  nop \\$*
.  ft
..
Our film class watched
.I The Effect of Gamma Rays on Man-in-the-Moon
Marigolds. \" whoops, the input line wrapped
.br
.I My own opus begins on line \n[.c] \
and ends on line \n[.c].
```

⇒ Our film class watched *The Effect of Gamma Rays on*
⇒ *Man-in-the-Moon* Marigolds.
⇒ *My own opus begins on line 11 and ends on line 12.*

\c [Escape sequence]
\n[.int] [Register]

\c continues an output line. Nothing on the input line after it is formatted. In contrast to \RET, a line after \c is treated as a new input line, so a control character is recognized at its beginning. The visual results depend on whether filling is enabled; see Section 5.9 [Manipulating Filling and Adjustment], page 101.

- If filling is enabled, 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.

- If filling is disabled, 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.
```

An intervening control line that causes a break overrides `\c`, flushing out the pending output line in the usual way.

The `.int` register contains a positive value if the last output line was continued with `\c`; this datum is associated with the environment (see Section 5.31 [Environments], page 204).[45]

## 5.17 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 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 5.28 [Traps], page 187, 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). The delimiter need not be a neutral apostrophe: `tl` accepts the same delimiters as most escape

[45] Historically, the `\c` escape sequence has proven challenging to characterize. Some sources say it "connects the next input text" (to the input line on which it appears); others describe it as "interrupting" text, on the grounds that a text line is interrupted without breaking, perhaps to inject a request invocation or macro call.

sequences; see Section 5.6.5 [Delimiters], page 91. Without an argument, `tl` is ignored.

- 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 collected 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.

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

The title line is printed using its own line length, which is specified (or incremented or decremented) with the `lt` request. Initially, the title line length is set to 6.5 i. If a negative line length is specified (which is not allowed), `gtroff` emits a warning in category '`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 is available in the `.lt` read-only register; it is associated with the environment (see Section 5.31 [Environments], page 204).

`.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 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 register `%`.

See Section 5.28 [Traps], page 187.

## 5.18 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 5.17 [Page Layout], page 128. This request is incorrectly documented in the AT&T `troff` manual as having a default scaling indicator of '`v`'.

```
.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 5.29 [Diversions], page 197).

The writable register `%` holds the current page number.

The register `.pe` is set to 1 while `bp` is active. See Section 5.28.1.1 [Page Location Traps], page 188.

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

Your text may *need* a certain amount of vertical space before a page break occurs. For instance, you may wish to ensure that the first output line of a paragraph is not *orphaned* at the bottom of a page. The `ne` request tests the amount of distance to the next page location trap (or the page bottom if none is planted earlier; see Section 5.28.1.1 [Page Location Traps], page 188), and breaks the page if less than *space* is available. The default scaling unit is '`v`'. If *space* is not specified, '`1v`' is assumed.

For example, to require room for at least the first two output lines of a paragraph, you can do the following.

```
.ne 2v
Considering how common illness is,
how tremendous the spiritual change that it brings,
how astonishing,
when the lights of health go down,
the undiscovered countries that are then disclosed,
```

This method is reliable only if no output line is pending when `ne` is invoked. When macro packages are used, this is often not the case: their paragraphing macros perform the break. You may need to experiment with placing the `ne` after the paragraphing macro, or `br` and `ne` before it.

`ne` is also useful to force grouping of section headings with their subsequent paragraphs, or tables with their captions and/or explanations. Macro packages often use `ne` with diversions to implement keeps and

displays; see Section 5.29 [Diversions], page 197. They may also offer parameters for widow and orphan management.

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

The `sv` request is similar to `ne`, but *saves* the specified vertical space. If *space* is available before the next page location trap (or the page bottom if none is planted earlier; see Section 5.28.1.1 [Page Location Traps], page 188), the space is output immediately. Any partially collected line is ignored. Otherwise, the vertical space is saved. Output the space on demand with `os`. Both `sv` and `os` ignore no-space mode (recall Section 5.11 [Manipulating Spacing], page 114). While the `sv` request allows negative values for *space*, `os` ignores them. The default scaling unit is '`v`'. If *space* is not specified, '`1v`' is assumed.

`\n[nl]` [Register]

`nl` interpolates or sets the vertical drawing position. When the formatter starts and first page transition hasn't happened yet, `nl` is negative. If a header trap has been planted on the page (typically at vertical position `0`), you can assign a negative value to `nl` to spring it if that page has already started (see Section 5.28.1.1 [Page Location Traps], page 188).

```
.de header
.  sp
.  tl ''Goldbach Solution''
.  sp
..
.
First page.
...
.bp
.wh 0 header \" plant header trap at top of page
.nr nl (-1)
Second page.
...
    ⇒ First page.
    ⇒
    ⇒ ...
    ⇒
    ⇒                    Goldbach Solution
    ⇒
    ⇒ Second page.
    ⇒
    ⇒ ...
```

Without resetting `nl` to a negative value, the trap just planted would be active beginning with the *next* page, not the current one.

See Section 5.29 [Diversions], page 197, for a comparison of `nl` with the `.h` and `.d` registers.

## 5.19 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).

### 5.19.1 Changing Fonts

`.ft` [*font*] [Request]
`\f`*f* [Escape sequence]
`\f(`*fn* [Escape sequence]
`\f[`*font*`]` [Escape sequence]
`\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 an escape sequence. The old syntax forms `\fP` or `\f[P]` are also supported.

Fonts are generally specified as uppercase 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 string-valued register '`.sty`'; it is associated with the environment (see Section 5.31 [Environ-

ments], page 204). If the current font isn't a style, interpolating '.sty' produces nothing.

See Section 5.19.3 [Font Positions], page 135, 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
     error  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 register '.zoom', in multiples of 1/1000th. It returns zero if there is no magnification.

### 5.19.2 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 sequence]
`\F(`*fm* [Escape sequence]
`\F[`*family*`]` [Escape sequence]
`\n[.fn]` [Register]

Set the font family to *family* (one-character name *f*, two-character name *fm*). If no argument is given, switch to the previous font family, or the default family if there is none. Use '`\F[]`' to do this with an escape sequence; '`\FP`' selects font family '`P`' instead. The initial font family is '`T`' (Times), but can be overridden by the output device description file—See Section 6.2.1 [DESC File Format], page 244. The current font family is available in the read-only string-valued register `.fam`; it is associated with the environment (see Section 5.31 [Environments], page 204).

```
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 read-only string-valued register `.fn` contains the current *real font name* of the current font. 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
    error  warning: can't find font 'FooR'
```

The default font position at startup 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 5.19.3 [Font Positions], page 135.

### 5.19.3 Font Positions

To support typeface indirection through styles, and for compatibility with AT&T `troff`, the formatter maintains a list of font *positions* at which fonts required by a document are *mounted*. An output device's description file `DESC` typically configures a set of pre-mounted fonts; see Section 6.2 [Device and Font Description Files], page 244. A font need not be explicitly mounted before it is selected; GNU `troff` will search `GROFF_FONT_PATH` for it by name and mount it at the first free mounting position on demand.

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

Mount the font named *font* at position *pos*, a non-negative integer. This numeric index can then be referred to with font selection instructions. When the formatter starts up, it reads the output device's description to mount an initial set of faces, and selects font position 1. Position 0

is unused by default. Unless the *external-name* argument is given, the name *font* should be that of a font description file stored in a directory corresponding to the selected output device.

The position of the currently selected font (or style) is available in the read-only register '`.f`'. It is associated with the environment (see Section 5.31 [Environments], page 204).

The value of `.f` can be copied to another register to save the current font or style for later recall.

```
.nr saved-font \n[.f]
.ft B
... text text text ...
.ft \n[saved-font]
```

The index of the next (nonzero) free font position is available in the read-only register '`.fp`'. Fonts not listed in the `DESC` file are automatically mounted at position '`\n[.fp]`' when selected. When mounting a font at a position explicitly with the `fp` request, this same practice should be followed, although GNU `troff` does not enforce this strictly.

If there is no third argument, the font description is sought in the file system under the name *font*.

The optional third argument *external-name* is the name of a font description file in the operating environment. GNU `troff` refuses to traverse directories to locate the font description. The second argument *font* is then the internal name of the font, which identifies it to GNU `troff` once it has been mounted. This feature enables font names to be aliased, which can be necessary in compatibility mode since AT&T `troff` syntax affords no means of identifying fonts with names longer than two characters, like '`TBI`' or '`ZCMI`', in a font selection escape sequence. See Section 5.38.2 [Compatibility Mode], page 225. You can also alias fonts on mounting for convenience or abstraction.

```
.ft SC ZCMI \" "script" font
Send a \f(SChand-written\fP thank-you note.
.ft Emph I
.ft Strong B
Are \f[Emph]these names\f[] \f[Strong]comfortable\f[]?
```

The `ft` request and `\f` escape sequence accept mounting positions in the place of font or style names.

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

Select font position *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 sequence, 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 5.19.1 [Changing Fonts], page 132, for font selection by name.

### 5.19.4 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 6.2 [Device and Font Description Files], page 244, and Section 5.19.6 [Special Fonts], page 145, 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 the `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;[46] 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 sequence]
\[*name*] [Escape sequence]
\[*base-glyph combining-component . . .*] [Escape sequence]

Typeset a special character *name* (two-character name *nm*) or a composite glyph consisting of *base-glyph* overlaid with one or more *combining-component*s. For example, '\[A ho]' is a capital letter "A" with a "hook accent" (ogonek).

There is no special syntax for one-character names—the analogous form '*n*' would collide with other escape sequences. However, the four escape sequences \', \-, _, and \`, are translated on input to the special character escape sequences \[aa], \[-], \[ul], and \[ga], respectively.

A special character name of length one is not the same thing as an ordinary input character: that is, the character a is not the same as \[a].

If *name* is undefined, a warning in category 'char' is produced and the escape is ignored. See Section 5.37.1 [Warnings], page 222, for information about the enablement and suppression of warnings.

GNU troff 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.

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

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 sequence]

Typeset the glyph named *xxx*.[47] Normally it is more convenient to use `\[`*xxx*`]`, but `\C` has the advantage that it is compatible with newer versions of AT&T `troff` and is available in compatibility mode.

`.composite` *from to* [Request]

Map glyph name *from* to glyph name *to* if it is used in `\[...]` with more than one component. See above for examples.

This mapping is based on glyph names only; no check for the existence of either glyph is done.

A set of default mappings for many accents can be found in the file `composite.tmac`, which is loaded at startup.

`\N'`*n*`'` [Escape sequence]

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'
```

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

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 sequence]
A backslash followed by the apostrophe character, ASCII character `0x27` (EBCDIC character `0x7D`), is a synonym for `\[aa]`, the acute accent.

`\`` [Escape sequence]
A backslash followed by ASCII character `0x60` (EBCDIC character `0x79` [usually]), is a synonym for `\[ga]`, the grave accent.

`\-` [Escape sequence]
A backslash followed by a dash is a synonym for `\[-]`, the minus sign.

`\_` [Escape sequence]
A backslash followed by an underscore is a synonym for `\[ul]`, the underrule. On `troff`-mode (typesetter) devices it may be font-invariant and drawn lower than the underscore glyph '`_`'.

`.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.[48] These properties can be 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 5.19.5 [Character Classes], page 144.

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.

[48] 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.

`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 numeric 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` *c* [*contents*] [Request]
`.fchar` *c* [*contents*] [Request]
`.fschar` *f c* [*contents*] [Request]
`.schar` *c* [*contents*] [Request]

Define a new character or glyph *c* to be *contents*, which can be empty. More precisely, `char` defines a `groff` object (or redefines an existing one) that is accessed with the name *c* on input, and produces *contents* on output. Every time glyph *c* needs to be printed, *contents* 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 *contents* is processed. Any emboldening, constant spacing, or track kerning is applied to this object rather than to individual glyphs in *contents*.

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 *c* 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 5.19.5 [Character Classes], page 144.

`.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*.

### 5.19.5 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 5.19.4 [Using Symbols], page 137, for more details.

### 5.19.6 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 5.19.4 [Using Symbols], page 137, 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 5.19.4 [Using Symbols], page 137, for the exact search order of glyphs.

### 5.19.7 Artificial Fonts

There are a number of requests and escape sequences 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 sequence]
`\H'+`*height*`'` [Escape sequence]
`\H'-`*height*`'` [Escape sequence]
`\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 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 type 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 sequence]
`\n[.slant]` [Register]

Slant the current font by *slant* degrees. Positive values slant to the right. Only integer values are possible.

The read-only 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 escape is incorrectly documented in the AT&T `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 quantity 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 environment (see Section 5.31 [Environments], page 204). 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.

### 5.19.8 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 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 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 6.2.2 [Font Description File Format], page 247.

*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 type 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 type size is greater than or equal to *s1* and less than or equal to *s2* the increase in width is a linear function of the type size.

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. Two escape sequences help with this.

\/ [Escape sequence]

Apply an *italic correction*: modify the spacing of the preceding glyph so that the distance between it and the following glyph is correct if the latter is of upright shape. For example, if an italic 'f' is followed immediately by a roman right parenthesis, then in many fonts the top right portion of the 'f' overlaps the top left of the right parenthesis, which is ugly. Use this escape sequence whenever an oblique glyph is immediately followed by an upright glyph without any intervening space.

\, [Escape sequence]

Apply a *left italic correction*: modify the spacing of the following glyph so that the distance between it and the preceding glyph is correct if the latter is of upright shape. For example, if a roman left parenthesis is immediately followed by an italic 'f', then in many fonts the bottom left portion of the 'f' overlaps the bottom of the left parenthesis, which is ugly. Use this escape sequence whenever an upright glyph is followed immediately by an oblique glyph without any intervening space.

### 5.19.9 Dummy Characters

As discussed in Section 5.1.7 [Requests and Macros], page 69, the first character on an input line is treated specially. Further, formatting a glyph has many consequences on formatter state (see Section 5.31 [Environments], page 204). Occasionally, we want to escape this context or embrace some of those consequences without actually rendering a glyph to the output.

\& [Escape sequence]

Interpolate a dummy character, which is constitutive of output but invisible.[49] Its presence alters the interpretation context of a subsequent input character, and enjoys several applications.

[49] Opinions of this escape sequence's name abound. "Zero-width space" is a popular misnomer: `roff` formatters do not treat it like a space. Ossanna called it a "non-printing, zero-width character", but the character causes *output* even though it does not "print". If no output line is pending, the dummy character starts one. Contrast an empty input document with one containing only `\&` (and a newline). The former produces no output; the latter, a blank page.

- It prevents insertion of extra space after an end-of-sentence character (see Section 5.1.2 [Sentences], page 66).

```
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 permits the `tr` request to remap a character to nothing (see Section 5.13 [Character Translations], page 121).

The dummy character escape sequence sees use in macro definitions as a means of ensuring that arguments are treated as text even if they begin with spaces or control characters.

```
.de HD \" typeset a simple bold heading
.  sp
.  ft B
\&\\$1 \" exercise: remove the \&
.  ft
.  sp
..
.HD .\|.\|.\|surprised?
```

One way to think about the dummy character is to imagine placing the symbol '`&`' in the input at a certain location; if doing so has all the side effects on formatting that you desire except for sticking an ugly ampersand in the midst of your text, the dummy character is what you want in its place.

`\)` [Escape sequence]

Interpolate a *transparent* dummy character—one that is transparent to end-of-sentence detection. It behaves as `\&`, except that `\&` is treated as letters and numerals normally are after '`.`', '`?`' and '`!`'; `\&` cancels end-of-sentence detection, and `\)` does not.

```
.de Suffix-&
.  nop \&\\$1
..
.
.de Suffix-)
.  nop \)\\$1
..
.
Here's a sentence.\c
.Suffix-& '
Another one.\c
.Suffix-) '
And a third.
    ⇒ Here's a sentence.' Another one.'  And a third.
```

## 5.20 Manipulating Type Size and Vertical Spacing

These concepts were introduced in Section 5.2 [Page Geometry], page 76. The height of a font's tallest glyph is one em, which is equal to the type size in points.[50] A vertical spacing of less than 120% of the type size can make a document hard to read. Larger proportions can be useful to spread the text for annotations or proofreader's marks. By default, GNU `troff` uses 10 point type on 12 point spacing. Typographers call the difference between type size and vertical spacing *leading*.[51]

### 5.20.1 Changing the Type Size

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

Use the `ps` request or the `\s` escape sequence to change (increase, decrease) the type size (in scaled points). Specify *size* as either an absolute type size, or as a relative change from the current size. `ps` with no argument restores the previous size. The `ps` request's default scaling unit is '`z`'. If the requested size is non-positive, it is set to 1 u.

The read-only string-valued register `.s` interpolates the type size in points as a decimal fraction; it is associated with the environment (see Section 5.31 [Environments], page 204). To obtain the type size in scaled

[50] In text fonts, the tallest glyphs are typically parentheses. Unfortunately, in many cases the actual dimensions of the glyphs in a font do not closely match its declared type size! For example, in the standard PostScript font families, 10-point Times sets better with 9-point Helvetica and 11-point Courier than if all three were used at 10 points.

[51] Pronounce "leading" to rhyme with "sledding"; it refers to the use of lead metal (Latin: *plumbum*) in traditional typesetting.

points, interpolate the `.ps` register instead (see Section 5.20.3 [Using Fractional Type Sizes], page 154).

```
snap, snap,
.ps +2
grin, grin,
.ps +2
wink, wink, \s+2nudge, nudge,\s+8 say no more!
.ps 10
```

The `\s` escape sequence supports a variety of syntax forms.

`\sn` Set the type 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 type size by *n* points. *n* must be exactly one digit.

`\s(nn` Set the type size to *nn* points. *nn* must be exactly two digits.

`\s+(nn`
`\s-(nn`
`\s(+nn`
`\s(-nn` Increase or decrease the type size by *nn* points. *nn* must be exactly two digits.

See Section 5.20.3 [Using Fractional Type Sizes], page 154, for further syntactical forms of the `\s` escape sequence that additionally accept decimal fractions.

The `\s` escape sequence affects the environment immediately and doesn't produce an input token. Consequently, it can be used in requests like `mc`, which expects a single character as an argument, to change the type size on the fly.

```
.mc \s[20]x\s[0]
```

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

Some devices may permit only certain type sizes, in which case GNU `troff` rounds to the nearest permissible size. The `DESC` file normally specifies which type sizes are allowed by the device.

Use the `sizes` request to change the permissible sizes for the output device. Arguments are in scaled points; See Section 5.20.3 [Using Fractional Type Sizes], page 154. Each can be a single type size (such as '`12000`'), or a range of sizes (such as '`4000-72000`'). You can optionally end the list with a zero.

### 5.20.2 Changing the Vertical Spacing

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

Change (increase, decrease) the vertical spacing by *space*. The default scaling unit is 'p'. If `vs` is called without an argument, the vertical spacing is reset to the previous value before the last call to `vs`. GNU `gtroff` emits a warning in category '`range`' if *space* is negative; the vertical spacing is then set to the smallest possible positive value, the vertical motion quantum (as found in the `.V` register).

'`.vs 0`' isn't saved in a diversion since it doesn't result in a vertical motion. You must explicitly issue this request before calling the diversion.

The read-only register `.v` contains the vertical spacing; it is associated with the environment (see Section 5.31 [Environments], page 204).

When a break occurs, GNU `troff` performs the following procedure.

- Move the drawing position vertically by the *extra pre-vertical line space*, the minimum of all negative `\x` escape sequence arguments in the pending output line.
- Move the drawing position vertically by the vertical line spacing.
- Write out the pending output line.
- Move the drawing position vertically by the *extra post-vertical line space*, the maximum of all positive `\x` escape sequence arguments in the line that has just been output.
- Move the drawing position vertically by the *post-vertical line spacing* (see below).

Prefer `vs` or `pvs` over `ls` to produce double-spaced documents. `vs` and `pvs` have finer granularity than `ls`; moreover, some preprocessors assume single spacing. See Section 5.11 [Manipulating Spacing], page 114, regarding the `\x` escape sequence 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 unit 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`. GNU `troff` emits a warning in category '`range`' if *space* is negative; the post-vertical spacing is then set to zero.

The read-only register `.pvs` contains the post-vertical spacing; it is associated with the environment (see Section 5.31 [Environments], page 204).

### 5.20.3 Using Fractional Type Sizes

AT&T `troff` interpreted all type size measurements in points. Combined with integer arithmetic, this design choice made it impossible to support, for instance, ten and a half-point type. In GNU `troff`, an output device can select a scaling factor that subdivides a point into "scaled points". A type size expressed in scaled points can thus represent a non-integral type size.

A *scaled point* is equal to 1/*sizescale* points, where *sizescale* is specified in the device description file `DESC`, and defaults to 1.[52] Requests and escape sequences in GNU `troff` interpret arguments that represent a type size in scaled points, which the formatter multiplies by *sizescale* and converts to an integer. Arguments treated in this way comprise those to the escape sequences `\H` and `\s`, to the request `ps`, the third argument to the `cs` request, and the second and fourth arguments to the `tkf` request. Scaled points may be specified explicitly with the `z` scaling unit.

For example, if *sizescale* is 1000, then a scaled point is one thousandth of a point. The request '`.ps 10.5`' is synonymous with '`.ps 10.5z`' and sets the type size to 10,500 scaled points, or 10.5 points. Consequently, in GNU `troff`, the register `.s` can interpolate a non-integral type size.

`\n[.ps]` [Register]
: This read-only register interpolates the type size in scaled points; it is associated with the environment (see Section 5.31 [Environments], page 204).

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

Another GNU `troff` scaling unit, '`s`', multiplies by the number of basic units in a scaled point. Thus, '`\n[.ps]s`' is equal to '`1m`' by definition. Do not confuse the '`s`' and '`z`' scaling units.

`\n[.psr]` [Register]
`\n[.sr]` [Register]
: Output devices may be limited in the type sizes they can employ. The `.s` and `.ps` registers represent the type size selected by the output driver as it understands a device's capability. The last *requested* type size is interpolated in scaled points by the read-only register `.psr` and in points as a decimal fraction by the read-only string-valued register `.sr`. Both are associated with the environment (see Section 5.31 [Environments], page 204).

[52] See Section 6.2 [Device and Font Description Files], page 244.

For example, if a type size of 10.95 points is requested, and the nearest size permitted by a `sizes` request (or a `sizescale` directive in the device's `DESC` file) is 11 points, the latter value is used by the output driver.

The `\s` escape sequence offers the following syntax forms that work with fractional type sizes and accept scaling units. You may of course give them integral arguments. The delimited forms need not use the neutral apostrophe; see Section 5.6.5 [Delimiters], page 91.

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

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

## 5.21 Colors

GNU `troff` supports color output with a variety of color spaces and up to 16 bits per channel. Some devices, particularly terminals, may be more limited. When color support is enabled, two colors are current at any given time: the *stroke color*, with which glyphs, rules (lines), and geometric objects like circles and polygons are drawn, and the *fill color*, which can be used to paint the interior of a closed geometric figure.

`.color` [*n*] [Request]
`\n[.color]` [Register]
If *n* is missing or non-zero, enable the output of color-related device-independent output commands (this is the default); otherwise, disable them. This request sets a global flag; it does not produce an input token (see Section 5.36 [Gtroff Internals], page 217).

The read-only register `.color` is 1 if colors are enabled, 0 otherwise.

Color can also be disabled with the `-c` command-line option.

`.defcolor` *ident scheme color-component ...* [Request]
Define a color named *ident*. *scheme* selects a color space and determines the quantity of required *color-component*s; it must be one of '`rgb`' (three components), '`cmy`' (three), '`cmyk`' (four), or '`gray`' (one). '`grey`' is accepted as a synonym of '`gray`'. The color components can be encoded as a single hexadecimal value starting with '`#`' or '`##`'. The former indicates

that each component is in the range 0–255 (0–FF), the latter the range 0–65,535 (0–FFFF).

```
.defcolor half gray #7f
.defcolor pink rgb #FFC0CB
.defcolor magenta rgb  ##ffff0000ffff
```

Alternatively, each color component can be specified as a decimal fraction in the range 0–1, interpreted using a default scaling indicator of `f`, which multiplies its value by 65,536 (but clamps it at 65,535).

```
.defcolor gray50 rgb 0.5 0.5 0.5
.defcolor darkgreen rgb 0.1f 0.5f 0.2f
```

Each output device has a color named '`default`', which cannot be redefined. A device's default stroke and fill colors are not necessarily the same. For the `dvi`, `html`, `pdf`, `ps`, and `xhtml` output devices, GNU `troff` automatically loads a macro file defining many color names at startup. By the same mechanism, the devices supported by `grotty` recognize the eight standard ISO 6429/EMCA-48 color names.[53]

`.gcolor` [*color*] [Request]
`\m`*c* [Escape sequence]
`\m(`*co* [Escape sequence]
`\m[`*color*`]` [Escape sequence]
`\n[.m]` [Register]

Set the stroke color to *color*.

```
.gcolor red
The next words
.gcolor
\m[red]are in red\m[]
and these words are in the previous color.
```

The escape sequence `\m[]` restores the previous stroke color, as does a `gcolor` request without an argument.

The name of the current stroke color is available in the read-only string-valued register '`.m`'; it is associated with the environment (see Section 5.31 [Environments], page 204). It interpolates nothing when the stroke color is the default.

`\m` doesn't produce an input token in GNU `troff` (see Section 5.36 [Gtroff Internals], page 217). It therefore 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[]
```

[53] also known vulgarly as "ANSI colors"

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

Set fill color for objects drawn with `\D'...'` escape sequences.

Create an ellipse with a red interior as follows.

```
\M[red]\h'0.5i'\D'E 2i 1i'\M[]
```

The escape sequence `\M[]` restores the previous fill color, as does an `fcolor` request without an argument.

The name of the current fill color is available in the read-only string-valued register '`.M`'; it is associated with the environment (see Section 5.31 [Environments], page 204). It interpolates nothing when the fill color is the default.

`\M` doesn't produce an input token in GNU `troff`.

## 5.22 Strings

GNU `troff` supports strings primarily for user convenience. Conventionally, if one would define a macro only to interpolate a small amount of text, without invoking requests or calling any other macros, one defines a string instead. Only one string is predefined by the language.

`\*[.T]` [String]

Contains the name of the output device (for example, '`utf8`' or '`pdf`').

The `ds` request creates a string with a specified name and contents and the `\*` escape sequence dereferences its name, interpolating its contents. If the string named by the `\*` escape sequence does not exist, it is defined as empty, nothing is interpolated, and a warning in category '`mac`' is emitted. See Section 5.37.1 [Warnings], page 222, for information about the enablement and suppression of warnings.

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

Define a string called *name* with contents *contents*. If *name* already exists, it is removed first (see `rm` below). If `ds` is called with only one argument, *name* is defined as an empty string.

The `\*` escape sequence interpolates a previously defined string variable *name* (one-character name *n*, two-character name *nm*). The bracketed interpolation form accepts arguments that are handled as macro arguments are; recall Section 5.6.3 [Calling Macros], page 88. In contrast to macro

calls, however, if a closing bracket '`]`' occurs in a string argument, that argument must be enclosed in double quotes. `\*` is interpreted even in copy mode (see Section 5.24.2 [Copy Mode], page 174). When defining strings, argument interpolations must be escaped if they are to reference parameters from the calling context; See Section 5.24.1 [Parameters], page 172.

```
.ds cite (\\$1, \\$2)
Gray codes are explored in \*[cite Morgan 1998].
    ⇒ Gray codes are explored in (Morgan, 1998).
```

**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 H2O H\v'+.3m'\s'-2'2\v'-.3m'\s00 \" water
```

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

```
.ds H2O H\v'+.3m'\s'-2'2\v'-.3m'\s00\" water
```

Ending string definitions (and appendments) with a comment, even an empty one, prevents unwanted space from creeping into them during source document maintenance.

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

An initial neutral double quote `"` in *contents* is stripped to allow embedding of leading spaces. Any other `"` is interpreted literally, but it is wise to use the special character escape sequence `\[dq]` instead if the string might be interpolated as part of a macro argument; see Section 5.6.3 [Calling Macros], page 88.

```
.ds salutation "         Yours in a white wine sauce,\"
.ds c-var-defn "  char mydate[]=\[dq]2020-07-29\[dq];\"
```

Strings are not limited to a single input line of text. `\RET` works just as it does elsewhere. The resulting string is stored *without* the newlines. Care is therefore required when interpolating strings while filling is disabled.

```
.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 `\*` to interpolate a macro instead; see Section 5.30 [Punning Names], page 202.

Because strings are similar to macros, they too can be defined so as to suppress AT&T `troff` compatibility mode when used; see Section 5.24 [Writing Macros], page 168, and Section 5.38.2 [Compatibility Mode], page 225. The `ds1` request defines a string such that compatibility mode is off when the string is later interpolated. To be more precise, a *com-*

*patibility 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: register '[' not defined
    ⇒ The value of xxx is 0xxx].
\*(bb
    ⇒ The value of xxx is 12345.
```

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

The `as` request is similar to `ds` but appends *contents* 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 the string).

```
.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 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 5.36 [Gtroff Internals], page 217, 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*, inclusively. 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 backward 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). Special characters in the string will often transform in the expected way due to the regular naming convention for accented characters. When they do not, use substrings and/or catenation.

```
.ds resume R\['e]sum\['e]
\*[resume]
.stringdown resume
\*[resume]
.stringup resume
\*[resume]
    ⇒ Résumé résumé RÉSUMÉ
```

(In practice, we would end the `ds` request with a comment escape `\"` to prevent space 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 in category '`mac`' is produced, and the request is ignored. See Section 5.37.1 [Warnings], page 222, for information about the enablement and suppression of warnings.

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 (probable infinite
error  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 5.24 [Writing Macros], page 168.

To remove an alias, call `rm` on its name. The object itself is not destroyed until it has no more names.

## 5.23 Conditionals and Loops

`groff` has `if` and `while` control structures like other languages. However, the syntax for grouping multiple input lines in the branches or bodies of these structures is unusual.

### 5.23.1 Operators in Conditionals

In `if`, `ie`, and `while` requests, in addition to the numeric expressions described in Section 5.4 [Numeric Expressions], page 79, several Boolean operators are available; the members of this expanded class are termed *conditional expressions*.

`c` *glyph* — True if *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 any of the `char`, `fchar`, `fschar`, or `schar` requests.

`d` *name* — True if a string, macro, diversion, or request called *name* exists.

`e` — True if the current page is even-numbered.

`F` *font*
: True if *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.

`m` *color*
: True if *color* is defined.

`n`
: True if the document is being processed in `nroff` mode. See Section 5.14 [`troff` and `nroff` Modes], page 123.

`o`
: True if the current page is odd-numbered.

`r` *register*
: True if *register* exists.

`S` *style*
: True if *style* is available for the current font family. Font translation is applied.

`t`
: True if the document is being processed in `troff` mode. See Section 5.14 [`troff` and `nroff` Modes], page 123.

`v`
: Always false. This condition is recognized only for compatibility with certain other `troff` implementations.[54]

`'`*xxx*`'`*yyy*`'`
: True if formatting the comparands *xxx* and *yyy* produces the same output commands. The delimiter need not be a neutral apostrophe: the output comparison operator accepts the same delimiters as most escape sequences; see Section 5.6.5 [Delimiters], page 91. This *output comparison operator* formats *xxx* and *yyy* in separate environments; after the comparison, the resulting data are discarded.

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

The resulting glyph properties, including font family, style, size, and slant, must match, but not necessarily the requests and/or escape sequences used to obtain them. In the previous example, '`|`' and '`\fR|\fP`' result in '`|`' glyphs in the same typefaces at the same positions, so the comparands are equal. If '`.ft I`' had been added before the '`.ie`', they would differ: the first '`|`' would produce an italic '`|`', not a roman one. Motions must match in orientation and magnitude to within the applicable horizontal and vertical motion quanta of the device, after rounding. '`.if "\u\d"\v'0'"`' is false even though both comparands result in

[54] 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.

zero net motion, because motions are not interpreted or optimized but sent as-is to the output.[55] On the other hand, '`.if "\d"\v'0.5m'"`' is true, because `\d` is defined as a downward motion of one-half em.[56]

Surround the comparands with `\?` to avoid formatting them; this causes them to be compared character by character, as with string comparisons in other programming languages.

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

Since comparands protected with `\?` are read in copy mode, they need not even be valid `groff` syntax. The escape character is still lexically recognized, however, and consumes the next character.

```
.ds a \[
.ds b \[
.if '\?\*a\?'\?\*b\?' a and b true
.if '\?\\?'\?\\?' backslash true \" doesn't work
    ⇒ a and b true
```

The above operators can't be combined with most others, but a leading '`!`', not followed immediately by spaces or tabs, complements an expression.

```
.nr x 1
.ie !r x register x is not defined
.el      register x is defined
    ⇒ register x is defined
```

Spaces and tabs are optional immediately after the '`c`', '`d`', '`F`', '`m`', '`r`', and '`S`' operators, but right after '`!`', they end the predicate and the conditional evaluates true.[57]

```
.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 '`r x`' in the output is a clue that our conditional was not interpreted as we planned, but matters may not always be so obvious.

---

[55] Because formatting of the comparands takes place in a dummy environment, vertical motions within them cannot spring traps.

[56] All of this is to say that the lists of output nodes created by formatting *xxx* and *yyy* must be identical. See Section 5.36 [Gtroff Internals], page 217.

[57] This bizarre behavior maintains compatibility with AT&T `troff`.

### 5.23.2 if-then

`.if` *cond-expr anything* [Request]

Evaluate the conditional expression *cond-expr*, and if it evaluates true (or to a positive value), interpret the remainder of the line *anything* as if it were an input line. Recall from Section 5.6.2 [Invoking Requests], page 86, that any quantity of spaces between arguments to requests serves only to separate them; leading spaces in *anything* are thus not seen. *anything* effectively *cannot* be omitted; if *cond-expr* is true and *anything* is empty, the newline at the end of the control line is interpreted as a blank input line (and therefore a blank text line).

```
super\c
tanker
.nr force-word-break 1
super\c
.if ((\n[force-word-break] = 1) & \n[.int])
tanker
    ⇒ supertanker super tanker
```

`.nop` *anything* [Request]

Interpret *anything* as if it were an input line. This is similar to '`.if 1`'. `nop` is not really "no operation"; its argument *is* processed—unconditionally. It can be used to cause text lines to share indentation with surrounding control lines.

```
.als real-MAC MAC
.de wrapped-MAC
.  tm MAC: called with arguments \\$@
.  nop \\*[real-MAC]\\
..
.als MAC wrapped-MAC
\# Later...
.als MAC real-MAC
```

In the above, we've used aliasing, `nop`, and the interpolation of a macro as a string to interpose a wrapper around the macro '`MAC`' (perhaps to debug it).

### 5.23.3 if-else

`.ie` *cond-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. Unusually among programming languages, any number of non-conditional requests may be interposed between the `ie` branch and the `el` branch.

```
.nr a 0
.ie \na a is nonzero.
.nr a +1
.el a was not positive but is now \na.
    ⇒ a was not positive but is now 1.
```

Another way in which `el` is an ordinary request is that it does not lexically "bind" more tightly to its `ie` counterpart than it does to any other request. This fact can surprise C programmers.

```
.nr a 1
.nr z 0
.ie \nz \
.  ie \na a is true
.  el     a is false
.el z is false
    error  warning: unbalanced 'el' request
    ⇒ a is false
```

To conveniently nest conditionals, keep reading.

### 5.23.4 Conditional Blocks

`\{` [Escape sequence]
`\}` [Escape sequence]

It is frequently desirable for a control structure to govern more than one request, macro call, text line, or a combination of the foregoing. The opening and closing brace escape sequences `\{` and `\}` perform such grouping; such *conditional blocks* can be nested. Brace escape sequences outside of control structures have no meaning and produce no output.

`\{` should appear (after optional spaces and tabs) immediately subsequent to the request's conditional expression. `\}` should appear on a line with other occurrences of itself as necessary to match `\{` sequences. It can be preceded by a control character, spaces, and tabs. Input after any quantity of `\}` sequences on the same line is processed only if all of the preceding conditions to which they correspond are true. Furthermore, a `\}` closing the body of a `while` request (discussed below) must be the last such escape sequence 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
```

The above behavior may challenge the intuition; it was implemented to retain compatibility with AT&T `troff`. For clarity, it is idiomatic to end input lines with `\{`, followed by `\RET` if desired to prevent the newline from being interpreted as a blank text line, and to precede `\}` on an input line with nothing more than a control character, spaces, tabs, and other instances of itself.

```
.de DEBUG
debug =
.ie \\$1 \{\
ON,
development
\}
.el \{\
OFF,
production
\}
version
..
.DEBUG 0
.br
.DEBUG 1
```

Try omitting the `\RET`s 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) the remainder of the input line is interpreted *as if it were on an input line by itself.*

We can use `ie`, `el`, and conditional blocks to simulate the multi-way "switch" or "case" control structures of other languages. The following example is adapted from the `groff man` package. Indentation is used to clarify the logic.

```
.\" Simulate switch/case in roff.
.      ie '\\$2'1' .ds title General Commands\"
.el \{.ie '\\$2'2' .ds title System Calls\"
.el \{.ie '\\$2'3' .ds title Library Functions\"
.el \{.ie '\\$2'4' .ds title Kernel Interfaces\"
.el \{.ie '\\$2'5' .ds title File Formats\"
.el \{.ie '\\$2'6' .ds title Games\"
.el \{.ie '\\$2'7' .ds title Miscellaneous Information\"
.el \{.ie '\\$2'8' .ds title System Management\"
.el \{.ie '\\$2'9' .ds title Kernel Development\"
.el                .ds title \" empty
.\}\}\}\}\}\}\}\}\}
```

### 5.23.5 while

groff provides a looping construct: the while request. Its syntax matches the if request.

.while *cond-expr anything* [Request]

Evaluate the conditional expression *cond-expr*, and repeatedly execute *anything* unless and until *cond-expr* evaluates false. *anything*, which is often a conditional block, is referred to as the while request's *body*.

```
.nr a 0 1
.while (\na < 9) \{\
\n+a,
.\}
\n+a
    ⇒ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
```

GNU troff treats the body of a while request similarly to that of a de request (albeit one not read in copy mode[58]), but stores it under an internal name and deletes it when the loop finishes. The operation of a macro containing a while request can slow significantly if the while body is large. Each time the macro is executed, the while body is parsed and stored again.

```
.de xxx
.  nr num 10
.  while (\\n[num] > 0) \{\
.    \" many lines of code
.    nr num -1
.  \}
..
```

An often better solution—and one that is more portable, since AT&T troff lacked the while request—is to instead write a recursive macro. It will be parsed only once.[59]

[58] See Section 5.24.2 [Copy Mode], page 174.

[59] unless you redefine it

```
.de yyy
.  if (\\n[num] > 0) \{\
.    \" many lines of code
.    nr num -1
.    yyy
.  \}
..
.
.de xxx
.  nr num 10
.  yyy
..
```

To prevent infinite loops, the default number of available recursion levels is 1,000 or somewhat less.[60] You can disable this protective measure, or raise the limit, by setting the `slimit` register. See Section 5.37 [Debugging], page 219.

As noted above, if a `while` body begins with a conditional block, its closing brace must end an input line.

```
.if 1 \{\
.  nr a 0 1
.  while (\n[a] < 10) \{\
.    nop \n+[a]
.\}\}
```

```
error  unbalanced brace escape sequences
```

`.break` [Request]

Exit a `while` loop. Do not confuse this request with a typographical break or the `br` request.

`.continue` [Request]

Skip the remainder of a `while` loop's body, immediately starting the next iteration.

## 5.24 Writing Macros

A *macro* is a stored collection of text and control lines that can be interpolated multiple times. Use macros to define common operations. Macros are called in the same way that requests are invoked. While requests exist for the purpose of creating macros, simply calling an undefined macro, or interpolating it as a string, will cause it to be defined as empty. See Section 5.5 [Identifiers], page 83.

`.de` *name* [*end*] [Request]

Define a macro *name*, replacing the definition of any existing request, macro, string, or diversion called *name*. GNU `troff` enters "copy mode",

[60] "somewhat less" because things other than macro calls can be on the input stack

storing subsequent input lines as the macro definition. If the optional second argument is not specified, the definition ends with the control line '..' (two dots). Alternatively, *end* identifies a macro whose call syntax at the start of a control line ends the definition of *name*; *end* is then called normally. A macro definition must end in the same conditional block (if any) in which it began (see Section 5.23.4 [Conditional Blocks], page 165). Spaces or tabs are permitted after the control character in the line containing this ending token (either '.' or '*end*'), but a tab immediately after the token prevents its recognition as the end of a macro definition. The macro *end* can be called with arguments.[61]

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
..
```

We can define one macro within another. Attempting to nest '..' naïvely will end the outer definition because the inner definition isn't interpreted as such until the outer macro is later interpolated. We can use an end macro instead. Each level of nesting should use a unique end macro.

An end macro need not be defined until it is called. This fact enables a nested macro definition to begin inside one macro and end inside another. Consider the following example.[62]

[61] While it is possible to define and call a macro '.', you can't use it as an end macro: during a macro definition, '..' is never handled as calling '.', even if '.de *name* .' explicitly precedes it.

[62] Its structure is adapted from, and isomorphic to, part of a solution by Tadziu Hoffman to the problem of reflowing text multiple times to find an optimal configuration for it. https://lists.gnu.org/archive/html/groff/2008-12/msg00006.html

```
.de m1
.  de m2 m3
you
..
.de m3
Hello,
Joe.
..
.de m4
do
..
.m1
know?
.  m3
What
.m4
.m2
    ⇒ Hello, Joe.  What do you know?
```

A nested macro definition *can* be terminated with '`..`' and nested macros *can* reuse end macros, but these control lines must be escaped multiple times for each level of nesting. The necessity of this escaping and the utility of nested macro definitions will become clearer when we employ macro parameters and consider the behavior of copy mode in detail.

`de` defines a macro that inherits the compatibility mode enablement status of its context (see Section 5.38 [Implementation Differences], page 224). Often it is desirable to make a macro that uses `groff` features callable from contexts where compatibility mode is on; for instance, when writing extensions to a historical macro package. To achieve this, compatibility mode needs to be switched off while such a macro is interpreted—without disturbing that state when it is finished.

`.de1` *name* [*end*] [Request]

The `de1` request defines a macro to be interpreted with compatibility mode disabled. When *name* is called, compatibility mode enablement status is saved; it is restored when the call completes. Observe the extra backlash before the interpolation of register '`xxx`'; we'll explore this subject in Section 5.24.2 [Copy Mode], page 174.

```
.nr xxx 12345
.de aa
The value of xxx is \\n[xxx].
.  br
..
.de1 bb
The value of xxx is \\n[xxx].
..
.cp 1
.aa
    error  warning: register '[' not defined
    ⇒ The value of xxx is 0xxx].
.bb
    ⇒ The value of xxx is 12345.
```

`.dei` *name* [*end*] [Request]
`.dei1` *name* [*end*] [Request]

The `dei` request defines a macro with its name and end macro indirected through strings. That is, it interpolates strings named *name* and *end* before performing the definition.

The following examples are equivalent.

```
.ds xx aa
.ds yy bb
.dei xx yy
.de aa bb
```

The `dei1` request bears the same relationship to `dei` as `de1` does to `de`; it temporarily turns compatibility mode off when *name* is called.

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

`am` appends subsequent input lines to macro *name*, extending its definition, and otherwise working as `de` does.

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 other requests are analogous to their '`de`' counterparts. The `am1` request turns off compatibility mode during interpretation of the appendment. The `ami` request appends indirectly, meaning that strings *name* and *end* are interpolated with the resulting names used before appending. The `ami1` request is similar to `ami`, disabling compatibility mode during interpretation of the appended lines.

Using `trace.tmac`, you can trace calls to `de`, `de1`, `am`, and `am1`. You can also use the `backtrace` request at any point desired to troubleshoot tricky spots (see Section 5.37 [Debugging], page 219).

See Section 5.22 [Strings], page 157, for the `als`, `rm`, and `rn` requests to create an alias of, remove, and rename a macro, respectively.

Macro identifiers share their name space with requests, strings, and diversions; see Section 5.5 [Identifiers], page 83. The `am`, `as`, `da`, `de`, `di`, and `ds` requests (together with their variants) create a new object only 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. See [the description of the `als` request], page 160, for pitfalls when redefining a macro that is aliased.

`.return` [*anything*] [Request]
Exit a macro, immediately returning to the caller. If called with an argument *anything*, exit twice—the current macro and the macro one level higher. This is used to define a wrapper macro for `return` in `trace.tmac`.

### 5.24.1 Parameters

Macro calls and string interpolations optionally accept a list of arguments; recall Section 5.6.3 [Calling Macros], page 88. At the time such an interpolation takes place, these *parameters* can be examined using a register and a variety of escape sequences starting with '`\$`'. All such escape sequences are interpreted even in copy mode, a fact we shall motivate and explain below (see Section 5.24.2 [Copy Mode], page 174).

`\n[.$]` [Register]
The count of parameters available to a macro or string is kept in this read-only register. The `shift` request can change its value.

Any individual parameter can be accessed by its position in the list of arguments to the macro call, numbered from left to right starting at 1, with one of the following escape sequences.

`\$`*n* [Escape sequence]
`\$(`*nn* [Escape sequence]
`\$[`*nnn*`]` [Escape sequence]
Interpolate the *n*th, *nn*th, or *nnn*th parameter. The first form expects only a single digit ($1 \le n \le 9$), the second two digits ($01 \le nn \le 99$), and the third any positive integer *nnn*. Macros and strings accept an unlimited number of parameters.

`.shift` [*n*] [Request]
Shift the parameters *n* places (1 by default). This is a "left shift": what was parameter *i* becomes parameter $i - n$. The parameters formerly in positions 1 to *n* are no longer available. Shifting by a nonpositive amount performs no operation. The register `.$` is adjusted accordingly.

In practice, parameter interpolations are usually seen prefixed with an extra escape character. This is because the \$ family of escape sequences is interpreted even in copy mode.[63]

\$* [Escape sequence]
\$@ [Escape sequence]
\$^ [Escape sequence]

In some cases it is convenient to interpolate all of the parameters at once (to pass them to a request, for instance). The \$* escape concatenates the parameters, separating them with spaces. \$@ is similar, concatenating the parameters, surrounding each with double quotes and separating them with spaces. If not in compatibility mode, the interpolation depth of double quotes is preserved (see Section 5.6.3 [Calling Macros], page 88). \$^ interpolates all parameters as if they were arguments to the ds request.

```
.de foo
. tm $1='\\$1'
. tm $2='\\$2'
. tm $*='\\$*'
. tm $@='\\$@'
. tm $^='\\$^'
..
.foo " This is a "test"
```

```
error  $1=' This is a '
error  $2='test"'
error  $*=' This is a  test"'
error  $@='" This is a " "test""'
error  $^='" This is a "test"'
```

\$* is useful when writing a macro that doesn't need to distinguish its arguments, or even to not interpret them; examples include macros that produce diagnostic messages by wrapping the tm or ab requests. Use \$@ when writing a macro that may need to shift its parameters and/or wrap a macro or request that finds the count significant. If in doubt, prefer \$@ to \$*. An application of \$^ is seen in trace.tmac, which redefines some requests and macros for debugging purposes.

\$0 [Escape sequence]

Interpolate the name by which the macro being interpreted was called. The als request can cause a macro to have more than one name. Applying string interpolation to a macro does not change this name.

[63] If they were not, parameter interpolations would be similar to command-line parameters—fixed for the entire duration of a roff program's run. The advantage of interpolating \$ escape sequences even in copy mode is that they can interpolate different contents from one call to the next, like function parameters in a procedural language. The additional escape character is the price of this power.

```
.de foo
.  tm \\$0
..
.als bar foo
.
.de aaa
.  foo
..
.de bbb
.  bar
..
.de ccc
\\*[foo]\\
..
.de ddd
\\*[bar]\\
..
.
.aaa
     error  foo
.bbb
     error  bar
.ccc
     error  ccc
.ddd
     error  ddd
```

### 5.24.2 Copy Mode

When GNU `troff` processes certain requests, most importantly those which define or append to a macro or string, it does so in *copy mode*: it copies the characters of the definition into a dedicated storage region, interpolating the escape sequences `\n`, `\g`, `\$`, `\*`, `\V`, and `\?` normally; interpreting `\RET` immediately; discarding comments `\"` and `\#`; interpolating the current leader, escape, or tab character with `\a`, `\e`, and `\t`, respectively; and storing all other escape sequences in an encoded form.

The complement of copy mode—a `roff` formatter's behavior when not defining or appending to a macro, string, or diversion—where all macros are interpolated, requests invoked, and valid escape sequences processed immediately upon recognition, can be termed *interpretation mode*.

`\\` [Escape sequence]

The escape character, `\` by default, can escape itself. This enables you to control whether a given `\n`, `\g`, `\$`, `\*`, `\V`, or `\?` escape sequence is

interpreted at the time the macro containing it is defined, or later when the macro is called.[64]

```
.nr x 20
.de y
.nr x 10
\&\nx
\&\\nx
..
.y
    ⇒ 20 10
```

You can think of \\ as a "delayed" backslash; it is the escape character followed by a backslash from which the escape character has removed its special meaning. Consequently, '\\' is not an escape sequence in the usual sense. In any escape sequence '*X*' that GNU troff does not recognize, the escape character is ignored and *X* is output. An unrecognized escape sequence causes a warning in category 'escape', with two exceptions—'\\' is the first.

\. [Escape sequence]

\. escapes the control character. It is similar to \\ in that it isn't a true escape sequence. It is used to permit nested macro definitions to end without a named macro call to conclude them. Without a syntax for escaping the control character, this would not be possible.

```
.de m1
foo
.
.  de m2
bar
\\..
.
..
.m1
.m2
    ⇒ foo bar
```

The first backslash is consumed while the macro is read, and the second is interpreted when macro m1 is called.

roff documents should not use the \\ or \. character sequences outside of copy mode; they serve only to obfuscate the input. Use \e to represent the escape character, \[rs] to obtain a backslash glyph, and \& before '.' and ''' where GNU troff expects them as control characters if you mean to use them literally (recall Section 5.1.7 [Requests and Macros], page 69).

Macro definitions can be nested to arbitrary depth. The mechanics of parsing the escape character have significant consequences for this practice.

[64] Compare this to the \def and \edef commands in TEX.

```
.de M1
\\$1
.  de M2
\\\\$1
.    de M3
\\\\\\\\$1
\\\\..
.    M3 hand.
\\..
.  M2 of
..
This understeer is getting
.M1 out
    ⇒ This understeer is getting out of hand.
```

Each escape character is interpreted twice—once in copy mode, when the macro is defined, and once in interpretation mode, when the macro is called. As seen above, this fact leads to exponential growth in the quantity of escape characters required to delay interpolation of `\n`, `\g`, `\$`, `\*`, `\V`, and `\?` at each nesting level, which can be daunting. GNU `troff` offers a solution.

`\E` [Escape sequence]

`\E` represents an escape character that is not interpreted in copy mode. You can use it to ease the writing of nested macro definitions.

```
.de M1
.  nop \E$1
.  de M2
.    nop \E$1
.    de M3
.      nop \E$1
\\\\..
.    M3 better.
\\..
.  M2 bit
..
This vehicle handles
.M1 a
    ⇒ This vehicle handles a bit better.
```

Observe that because `\.` is not a true escape sequence, we can't use `\E` to keep '`..`' from ending a macro definition prematurely. If the multiplicity of backslashes complicates maintenance, use end macros.

`\E` is also convenient to define strings containing escape sequences that need to work when used in copy mode (for example, as macro arguments), or which will be interpolated at varying macro nesting depths. We might define strings to begin and end superscripting as follows.[65]

[65] These are lightly adapted from the `groff` implementation of the `ms` macros.

```
.ds { \v'-.9m\s'\En[.s]*7u/10u'+.7m'
.ds } \v'-.7m\s0+.9m'
```

When the `ec` request is used to redefine the escape character, `\E` also makes it easier to distinguish the semantics of an escape character from the other meaning(s) its character might have. Consider the use of an unusual escape character, '`-`'.

```
.nr a 1
.ec -
.de xx
--na
..
.xx
    ⇒ -na
```

This result may surprise you; some people expect '`1`' to be output since register '`a`' has clearly been defined with that value. What has happened? The robotic replacement of '`\`' with '`-`' has led us astray. You might recognize the sequence '`--`' more readily with the default escape character as '`\-`', the special character escape sequence for the minus sign glyph.

```
.nr a 1
.ec -
.de xx
-Ena
..
.xx
    ⇒ 1
```

## 5.25 Page Motions

See Section 5.11 [Manipulating Spacing], page 114, for a discussion of the most commonly used request for vertical motion, `sp`, which spaces downward by one vee.

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

You can *mark* a location on a page for subsequent *return*. `mk` takes an argument, a register name in which to store the current page location. If given no argument, it stores the location in an internal register. This location can be used later by the `rt` or the `sp` requests (or the `\v` escape).

The `rt` request returns *upward* to the location marked with the last `mk` request. If used with an argument, it returns to a vertical position *dist* from the top of the page (no previous call to `mk` is necessary in this case). The default scaling indicator is '`v`'.

If a page break occurs between a `mk` request and its matching `rt` request, the `rt` request is silently ignored.

A simple implementation of a macro to set text in two columns follows.

```
.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
.  \}
..
```

Now let us apply our two-column macro.

```
.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.
    ⇒ 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.
```

Several escape sequences enable fine control of movement about the page.

\v'*expr*' [Escape sequence]
Vertically move the drawing position. *expr* indicates the magnitude of motion: positive is downward and and negative upward. The default scaling unit is 'v'. The motion is relative to the current drawing position unless *expr* begins with the boundary-relative motion operator '|'. See Section 5.4 [Numeric Expressions], page 79.

Text processing continues at the new drawing position; usually, vertical motions should be in balanced pairs to avoid a confusing page layout.

\v will not spring a vertical position trap. This can be useful; for example, consider a page bottom trap macro that prints a marker in the margin to indicate continuation of a footnote. See Section 5.28 [Traps], page 187.

A few escape sequences that produce vertical motion are unusual. They are thought to originate early in AT&T nroff history to achieve super- and subscripting by half-line motions on line printers and teletypewriters before the phototypesetter made more precise positioning available. They are reckoned in ems—not vees—to maintain continuity with their original purpose of moving relative to the size of the type rather than the distance between text baselines (vees).[66]

\r [Escape sequence]
\u [Escape sequence]
\d [Escape sequence]
Move upward 1 m, upward .5 m, and downward .5 m, respectively.

Let us see these escape sequences in use.

```
Obtain 100 cm\u3\d of \ka\d\092\h'|\nau'\r233\dU.
```

In the foregoing we have paired \u and \d to typeset a superscript, and later a full em negative ("reverse") motion to place a superscript above a subscript. A numeral-width horizontal motion escape sequence aligns the proton and nucleon numbers, while \k marks a horizontal position to which \h returns so that we could stack them. (We shall discuss these horizontal motion escape sequences presently.) In serious applications, we often want to alter the type size of the -scripts and to fine-tune the vertical motions, as the groff ms package does with its super- and subscripting string definitions.

\h'*expr*' [Escape sequence]
Horizontally move the drawing position. *expr* indicates the magnitude of motion: positive is rightward and negative leftward. The default scaling unit is 'm'. The motion is relative to the current drawing position unless *expr* begins with the boundary-relative motion operator '|'. See Section 5.4 [Numeric Expressions], page 79.

[66] At the grops defaults of 10-point type on 12-point vertical spacing, the difference between half a vee and half an em can be subtle: large spacings like '.vs .5i' make it obvious.

The following string definition sets the TEX logo.[67]

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

There are a number of special-case escape sequences for horizontal motion.

`\SP` [Escape sequence]
Move right one word space. (The input is a backslash followed by a space.) This escape sequence can be thought of as a non-adjustable, unbreakable space. Usually you want `\~` instead; see Section 5.9 [Manipulating Filling and Adjustment], page 101.

`\|` [Escape sequence]
Move one-sixth em to the right on typesetting output devices. If a glyph named '`\|`' is defined in the current font, its width is used instead, even on terminal output devices.

`\^` [Escape sequence]
Move one-twelfth em to the right on typesetting output devices. If a glyph named '`\^`' is defined in the current font, its width is used instead, even on terminal output devices.

`\0` [Escape sequence]
Move right by the width of a numeral in the current font.

Horizontal motions are not discarded at the end of an output line as word spaces are. See Section 5.1.4 [Breaking], page 68.

`\w'`*anything*`'` [Escape sequence]
`\n[st]` [Register]
`\n[sb]` [Register]
`\n[rst]` [Register]
`\n[rsb]` [Register]
`\n[ct]` [Register]
`\n[ssc]` [Register]
`\n[skw]` [Register]
Interpolate the width of *anything* in basic units. This escape sequence allows several properties of formatted output to be measured without writing it out.

```
The length of the string 'abc' is \w'abc'u.
    ⇒ The length of the string 'abc' is 72u.
```

*anything* is processed in a dummy environment: this means that font and type size changes, for example, may occur within it without affecting subsequent output.

[67] See Section 5.22 [Strings], page 157, for an explanation of the trailing '`\"`'.

After each use, `\w` sets several registers.

`st`
`sb` The maximum vertical displacements of the text baseline above and below, respectively. The sign convention is opposite that of relative vertical motions; that is, depth below the (original) baseline is negative. These registers are incorrectly documented in the AT&T `troff` manual as "the highest and lowest extent of [the argument to `\w`] relative to the baseline".

`rst`
`rsb` Like `st` and `sb`, but taking account of the heights and depths of glyphs. In other words, these registers store the highest and lowest vertical positions attained by *anything*, doing what AT&T `troff` documented `st` and `sb` as doing.

`ct` Characterizes the geometry of glyphs occurring in *anything*.

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 sequence]
`\k(`*ps* [Escape sequence]
`\k[`*position*`]` [Escape sequence]
Store the current horizontal position in the *input* line in a register with the 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 register containing the current horizontal output position (relative to the current indentation).

`\o'`*abc*`'` [Escape sequence]
Overstrike glyphs *a*, *b*, *c*, ...; the glyphs are centered, and the resulting spacing is the largest width of the affected glyphs.

`\z`*c* [Escape sequence]
Print glyph *c* with zero width, i.e., without spacing. Use this to overstrike glyphs left-aligned.

`\Z'`*anything*`'` [Escape sequence]
Save the drawing position, format *anything*, then restore it. The argument may not contain tabs or leaders.

An example of a strike-through macro follows.

```
.de ST
.nr ww \w'\\$1'
\Z@\v'-.25m'\l'\\n[ww]u'@\\$1
..
.
This is
.ST "a test"
an actual emergency!
```

## 5.26 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 5.25 [Page Motions], page 177), 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`.

All drawing is done via escape sequences.

`\l'`*l*`'` [Escape sequence]
`\l'`*lc*`'` [Escape sequence]
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 *c* 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 textbox
\[br]\\$*\[br]\l'|0\[rn]'\l'|0\[ul]'
..
```

The 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

escape sequences 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 sequence]
`\L'`*lg*`'` [Escape sequence]
Draw vertical lines. Its parameters are similar to the `\l` escape, except that the default scaling indicator is 'v'. The movement is downward for positive values, and upward for negative values. The default glyph is the box rule glyph, `\[br]`. As with the vertical motion escape sequences, 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 sequence]
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 TEXTBOX
.  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 motion quantum (the value of the `hor` keyword in the `DESC` file). Use `\M` (see Section 5.21 [Colors], page 155) 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 TEXTBOX
.  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* basic units. A value of zero selects the smallest available line thickness. A negative value makes the line thickness proportional to the current type 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 5.21 [Colors], page 155); 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 6.1.2.3 [Graphics Commands], page 236.

`\b'`*contents*`'` [Escape sequence]
Pile and center a sequence of glyphs vertically on the output line. *Piling* vertically stacks glyphs corresponding to each character in *contents*, read from left to right, and placed from top to bottom. GNU `troff` separates the glyphs vertically by 1 m, and the pile itself is centered 0.5 m above the text baseline. The horizontal drawing position is then advanced by the width of the widest glyph in the pile.

This rather inflexible positioning algorithm doesn't work with the `dvi` output device since its bracket pieces vary in height. Instead, use the `geqn` preprocessor.

See Section 5.11 [Manipulating Spacing], page 114, to see how to adjust the vertical spacing of the output line with the `\x` escape sequence.

The idiomatic use of `\b` is for building large brackets and braces, hence its name. We might construct a large opening brace as follows.

```
\b'\[lt]\[bv]\[lk]\[bv]\[lb]'
```

## 5.27 Deferring Output

A few roff language elements are generally not used in simple documents, but arise as page layouts become more sophisticated and demanding. *Environments* collect formatting parameters like line length and typeface. A *diversion* stores formatted output for later use. A *trap* is a condition on the input or output, tested automatically by the formatter, that is associated with a macro, causing it to be called when that condition is fulfilled.

Footnote support often exercises all three of the foregoing features. A simple implementation might work as follows. A pair of macros is defined: one starts a footnote and the other ends it. The author calls the first macro where a footnote marker is desired. The macro establishes a diversion so that the footnote text is collected at the place in the body text where its corresponding marker appears. An environment is created for the footnote so that it is set at a smaller typeface. The footnote text is formatted in the diversion using that environment, but it does not yet appear in the output. The document author calls the footnote end macro, which returns to the previous environment and ends the diversion. Later, after much more body text in the document, a trap, set a small distance above the page bottom, is sprung. The macro called by the trap draws a line across the page and emits the stored diversion. Thus, the footnote is rendered.

Diversions and traps make the text formatting process non-linear. Let us imagine a set of text lines or paragraphs labelled '`A`', '`B`', and so on. If we set up a trap that produces text '`T`' (as a page footer, say), and we also use a diversion to store the formatted text '`D`', then a document with input text in the order '`A B C D E F`' might render as '`A B C E T F`'. The diversion '`D`' will never be output if we do not call for it.

Environments of themselves are not a source of non-linearity in document formatting: environment switches have immediate effect. One could always write a macro to change as many formatting parameters as desired with a single convenient call. But because diversions can be nested and macros called by traps that are sprung by other trap-called macros, they may be called upon in varying contexts. For example, consider a page header that is always to be set in Helvetica. A document that uses Times for most of its body text, but Courier for displayed code examples, poses a challenge if a page break occurs in the middle of a code display; if the header trap assumes that the "previous font" is always Times, the rest of the example will be formatted in the wrong typeface. One could carefully save all formatting parameters upon entering the trap and restore them upon leaving it, but this is verbose, error-prone, and not future-proof as the groff language develops. Environments save us considerable effort.

## 5.28 Traps

*Traps* are locations in the output or conditions on the input that, when reached or fulfilled, call a specified macro. These traps can occur at a given

location on the page, at a given location in the current diversion (together, these are known as *vertical position traps*), at a blank line, at a line with leading space characters, after a quantity of input lines, or at the end of input. Macros called by traps are passed no arguments. Setting a trap is also called *planting* one. It is said that a trap is *sprung* if its condition is fulfilled.

### 5.28.1 Vertical Position Traps

*Vertical position traps* perform an action when GNU `troff` reaches or passes a certain vertical location on the output page or in a diversion. Their applications include setting page headers and footers, body text in multiple columns, and footnotes.

`.vpt` [*flag*] [Request]
`\n[.vpt]` [Register]
Enable vertical position traps if *flag* is non-zero or absent; disable them otherwise. Vertical position traps are those set by the `wh` request or by `dt` within a diversion. The parameter that controls whether vertical position traps are enabled is global. Initially, vertical position traps are enabled. The current value is stored in the `.vpt` read-only register.

A page can't be ejected if `vpt` is set to zero.

#### 5.28.1.1 Page Location Traps

`.wh` *dist* [*name*] [Request]
Call macro *name* when the vertical position *dist* on the page is reached or passed in the downward direction. The default scaling unit is '`v`'. 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. An existing *visible* trap (see below) at *dist* is removed; this is `wh`'s sole function if *name* is missing.

A trap is sprung only if it is *visible*, meaning that its location is reachable on the page[68] and it is not hidden by another trap at the same location already planted there.

An example of how a macro package might set headers and footers follows.

[68] A trap planted at '`20i`' or '`-30i`' will not be sprung on a page of length '`11i`'.

```
.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 above the top or at or below the bottom of the page can be made visible by either moving it into the page area or increasing the page length so that the trap is on the page. Negative trap values always use the *current* page length; they are not converted to an absolute vertical position. We can use the `ptr` request to dump our page location traps to the standard error stream (see Section 5.37 [Debugging], page 219). Their positions are reported in basic units appropriate to the device; an `nroff` device example follows.

```
.pl 5i
.wh -1i xx
.ptr
    error  xx       -240
.pl 100i
.ptr
    error  xx       -240
```

It is possible to have more than one trap at the same location (although only one at a time can be visible); to achieve this, the traps must be defined at different locations, then moved to the same place with the `ch` request. In the following example, the many empty lines caused by the `bp` request are not shown in the output.

```
.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]
The read-only register `.t` holds the distance to the next vertical position trap. If there are no traps between the current position and the bottom of the page, it contains the distance to the page bottom. Within a diversion, in the absence of a diversion trap, this distance is the largest representable integer in basic units—effectively infinite.

`.ch` *name* [*dist*] [Request]
Change the location of a trap by moving macro *name* to new location *dist*, or by unplanting it altogether if *dist* is absent. The default scaling unit is 'v'. Parameters to `ch` are specified in the opposite order from `wh`. If *name* is the earliest planted macro of multiple traps at the same location, (re)moving it from that location exposes the macro next least recently planted at the same place.[69]

Changing a trap's location is useful for building up footnotes in a diversion to allow more space at the bottom of the page for them.

The same macro can be installed simultaneously at multiple locations; however, only the earliest-planted instance—that has not yet been deleted with `wh`—will be moved by `ch`. The following example (using an `nroff`

[69] It may help to think of each trap location as maintaining a queue; `wh` operates on the head of the queue, and `ch` operates on its tail. Only the trap at the head of the queue is visible.

device) illustrates this behavior.[70] Blank lines have been elided from the output.

```
.de T
Trap sprung at \\n(nlu.
.br
..
.wh 1i T
.wh 2i T
foo
.sp 11i
.bp
.ch T 4i
bar
.sp 11i
.bp
.ch T 5i
baz
.sp 11i
.bp
.wh 5i
.ch T 6i
qux
.sp 11i
```

```
⇒ foo
⇒ Trap sprung at 240u.
⇒ Trap sprung at 480u.
⇒ bar
⇒ Trap sprung at 480u.
⇒ Trap sprung at 960u.
⇒ baz
⇒ Trap sprung at 480u.
⇒ Trap sprung at 1200u.
⇒ qux
⇒ Trap sprung at 1440u.
```

`\n[.ne]` [Register]

The read-only register `.ne` contains the amount of space that was needed in the last `ne` request that caused a trap to be sprung; it is useful in conjunction with the `.trunc` register. See Section 5.18 [Page Control], page 129.

Since the `.ne` register is set only by traps, it doesn't make sense to interpolate it outside of macros called by traps.

[70] . . . which is compatible with Heirloom Doctools `troff`.

`\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 set only by traps, it doesn't make sense to interpolate it outside of macros called by traps.

`\n[.pe]` [Register]
A read-only register containing 1 while a page is being ejected with the `bp` request (or by the end of input), and 0 otherwise.

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 calls 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.

### 5.28.1.2 Diversion Traps

`.dt` [*dist name*] [Request]
Set a trap *within* a diversion at location *dist*, which is interpreted relative to diversion rather than page boundaries. There exists only a single diversion trap per diversion. If invoked with fewer than two arguments, any diversion trap in the current diversion is removed. The register `.t` works within diversions. It is an error to invoke `dt` in the top-level diversion. See Section 5.29 [Diversions], page 197.

### 5.28.2 Input Line Traps

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

Set an input line trap, calling macro *name* after processing the next *n* productive input lines (see Section 5.9 [Manipulating Filling and Adjustment], page 101). Any existing input line trap in the environment is replaced. Without arguments, `it` and `itc` clear any input line trap that has not yet sprung.

Consider a macro '`.ST s n`' which sets the next *n* input lines in the font style *s*.

```
.de ST \" Use style $1 for next $2 text lines.
.  it \\$2 ES
.  ft \\$1
..
.de ES \" end ST
.  ft R
..
.ST I 1
oblique
face
.ST I 1
oblique\c
face
```

⇒ *oblique* face *oblique*face (second "face" upright)

Unlike the `ce` and `rj` requests, `it` counts lines interrupted with the `\c` escape sequence separately (see Section 5.16 [Line Continuation], page 127); `itc` does not. To see the difference, let's change the previous example to use `itc` instead.

```
...
.  itc \\$2 ES
...
```

⇒ *oblique* face *obliqueface* (second "face" oblique)

You can think of the `ce` and `rj` requests as implicitly creating an input line trap with `itc` that schedules a break when the trap is sprung.

```
.de BR
.  br
.  internal: disable centering-without-filling
..
.
.de ce
.  if \\n[.br] .br
.  itc \\$1 BR
.  internal: enable centering-without-filling
..
```

Let us consider in more detail the sorts of input lines that are or are not "productive".

```
.de Trap
TRAP SPRUNG
..
.de Mac
.if r a \l'5n'
..
.it 2 Trap
.
foo
.Mac
bar
baz
.it 1 Trap
.sp \" moves, but does not write or draw
qux
.itc 1 Trap
\h'5n'\c \" moves, but does not write or draw
jat
```

When '`Trap`' gets called depends on whether the '`a`' register is defined; the control line with the `if` request may or may not produce written output. We also see that the spacing request `sp`, while certainly affecting the output, does not spring the input line trap. Similarly, the horizontal motion escape sequence `\h` also affected the output, but was not "written". Observe that we had to follow it with `\c` and use `itc` to prevent the newline at the end of the text line from causing a word break, which, like an ordinary space character, counts as written output.

```
$ groff -Tascii input-trap-example.groff
    ⇒ foo bar TRAP SPRUNG baz
    ⇒
    ⇒ qux TRAP SPRUNG      jat TRAP SPRUNG
$ groff -Tascii -ra1 input-trap-example.groff
    ⇒ foo _____ TRAP SPRUNG bar baz
    ⇒
    ⇒ qux TRAP SPRUNG      jat TRAP SPRUNG
```

Input line traps are associated with the environment (see Section 5.31 [Environments], page 204); switching to another environment suspends the current input line trap, and going back resumes it, restoring the count of qualifying lines enumerated in that environment.

### 5.28.3 Blank Line Traps

`.blm` [*name*] [Request]
Set a blank line trap, calling the macro *name* when GNU `troff` encounters a blank line in an input file, instead of the usual behavior (see Section 5.1.4 [Breaking], page 68). A line consisting only of spaces is also treated as blank and subject to this trap. If no argument is supplied, the default blank line behavior is (re-)established.

### 5.28.4 Leading Space Traps

`.lsm` [*name*] [Request]
`\n[lsn]` [Register]
`\n[lss]` [Register]
Set a leading space trap, calling the macro *name* when GNU `troff` encounters leading spaces in an input line; the implicit line break that normally happens in this case is suppressed. If no argument is supplied, the default leading space behavior is (re-)established (see Section 5.1.4 [Breaking], page 68).

The count of leading spaces on an input line is stored in register `lsn`, and the amount of corresponding horizontal motion in register `lss`, irrespective of whether a leading space trap is set. When it is, the leading spaces are removed from the input line, and no motion is produced before calling *name*.

### 5.28.5 End-of-input Traps

`.em` [*name*] [Request]
Set a trap at the end of input, calling macro *name* after the last line of the last input file has been processed. If no argument is given, any existing end-of-input trap is removed.

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 (compatibility with AT&T `troff`), the end-of-input macro exits as soon as it causes a page break if no partially collected line remains.[71]

Let us assume that there is no `\c` in the above `approval` macro, that the page is full, and last output line has been broken with, say, a `br` request. Because there is no more room, a `ne` request at this point causes a page ejection, which in turn makes `troff` exit immediately as just described. In most situations, this is not desired; people generally want to format the input after `ne`.

To force processing of the whole end-of-input macro independently of this behavior, it is thus advisable to (invisibly) ensure the existence of a partially collected line (`\c`) whenever there is a chance that a page break can happen. In the above example, invoking the `ne` request ensures that there is room for the subsequent formatted output on the same page, so we need insert `\c` only once.

The next example shows how to append three lines, then start 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-of-input macro; further page breaks emitted by the macro are handled normally.

[71] While processing an end-of-input macro, the formatter assumes that the next page break must be the last; it goes into "sudden death overtime".

Another possible use of the em request is to make GNU troff emit a single large page instead of multiple pages. For example, one may want to produce a long plain text file for reading in a terminal or emulator without page footers and headers interrupting the body of the document. One approach is to set the page length at the beginning of the document to a very large value to hold all the text,[72] 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]: current vertical position
..
.
.de single-page-mode
.  pl 99999
.  em adjust-page-length
..
.
.\" Activate the above code if configured.
.if \n[do-continuous-rendering] \
.  single-page-mode
```

Since only one end-of-input trap exists and another macro package 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.

## 5.29 Diversions

In roff systems it is possible to format text as if for output, but instead of writing it immediately, one can *divert* the formatted text into a named storage area. It is retrieved later by specifying its name after a control character. The same name space is used for such *diversions* as for strings and macros; see Section 5.5 [Identifiers], page 83. Such text is sometimes said to be "stored in a macro", but this coinage obscures the important distinction between macros and strings on one hand and diversions on the other; the former store *unformatted* input text, and the latter capture *formatted* output. Diversions also do not interpret arguments. Applications of diversions include "keeps" (preventing a page break from occurring at an inconvenient place by forcing a set of output lines to be set as a group), footnotes, tables of contents, and indices. For orthogonality it is said that GNU troff is in the *top-level diversion* if no diversion is active (that is, formatted output is being "diverted" immediately to the output device).

Dereferencing an undefined diversion will create an empty one of that name and cause a warning in category 'mac' to be emitted. See Section 5.37.1 [Warnings], page 222, for information about the enablement and suppression

[72] Another, taken by the groff man macros, is to intercept ne requests and wrap bp ones.

of warnings. A diversion does not exist for the purpose of testing with the `d` conditional operator until its initial definition ends (see Section 5.23.1 [Operators in Conditionals], page 161). The following requests are used to create and alter diversions.

`.di` [*name*] [Request]
`.da` [*name*] [Request]
Start collecting formatted output in a diversion called *name*. The `da` request appends to the existing diversion called *name*, creating it if necessary. The pending output line is diverted as well. Switching to another (empty) environment (with the `ev` request) avoids the inclusion of the current partially collected line; Section 5.31 [Environments], page 204.

Invoking `di` or `da` without an argument stops diverting output to the diversion named by the most recent corresponding request. If `di` or `da` is called without an argument when there is no current diversion, a warning in category '`di`' is produced. See Section 5.37.1 [Warnings], page 222, for information about the enablement and suppression of warnings.

```
Before the diversion.
.di yyy
In the diversion.
.br
.di
After the diversion.
.br
    ⇒ After the diversion.
.yyy
    ⇒ Before the diversion.  In the diversion.
```

Because it is often desirable to exclude the partially collected line from a diversion, `groff` supports an alternative form of diversion known as a *box*.

`.box` [*name*] [Request]
`.boxa` [*name*] [Request]
Divert (or append) output to *name*, similarly to the `di` and `da` requests, respectively. Any output line pending when these requests are invoked is *not* included in the box. Calling `box` or `boxa` without an argument stops diverting output to the box named by the most recent corresponding request; a pending output line inside a diversion is discarded.

```
Before the box.
.box xxx
In the box.
.br
Hidden treasure.
.box
After the box.
.br
    ⇒ Before the box.  After the box.
.xxx
    ⇒ In the box.
```

Apart from pending output line inclusion and the request names that populate them, boxes are handled exactly as diversions are. All of the following `groff` language elements can be used with them interchangeably.

`\n[.z]` [Register]
`\n[.d]` [Register]

Diversions may be nested. The read-only string-valued register `.z` contains the name of the current diversion. The read-only register `.d` contains the current vertical place in the diversion. If the input text is not being diverted, `.d` reports the same location as the register `nl`.

`\n[.h]` [Register]

The read-only register `.h` stores 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.[73]

```
.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 implied by the example, vertical motion does not produce text baselines and thus does not increase the value interpolated by '`\n[.h]`'.

`\n[dn]` [Register]
`\n[dl]` [Register]

After completing a diversion, the writable registers `dn` and `dl` contain its vertical and horizontal sizes. Only the lines just processed are counted: for the computation of `dn` and `dl`, the requests `da` and `boxa` are handled as if `di` and `box` had been used, respectively—lines that have been already stored in the diversion (box) are not taken into account.

[73] Thus, the "water" gets "higher" proceeding *down* the page.

```
.\" Center text both horizontally and vertically.
.\" Macro .(c starts centering mode; .)c terminates it.
.
.\" Disable the escape character with .eo so that we
.\" don't have to double backslashes on the "\n"s.
.eo
.de (c
.  br
.  ev (c
.  evc 0
.  in 0
.  nf
.  di @c
..

.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
..
.ec
```

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

*Transparently* embed *anything* into the current diversion, preventing requests, macro calls, and escape sequences from being interpreted when read into a diversion. This is useful for preventing them from taking effect until the diverted text is actually output. The `\!` escape sequence transparently embeds input up to and including the end of the line. The `\?` escape sequence transparently embeds input until its own next occurrence.

*anything* may not contain newlines; use `\!` by itself 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 escape sequences read the data in copy mode.

If `\!` is used in the top-level diversion, its argument is directly embedded into GNU `troff`'s intermediate output. This can be used, for example, to control a postprocessor that processes the data before it is sent to an output driver.

The `\?` escape used in the top-level diversion produces no output at all; its argument is simply ignored.

`.output` *contents* [Request]

Emit *contents* directly to GNU `troff`'s intermediate output (subject to copy mode interpretation); this is similar to `\!` used at the top level. An initial neutral double quote in *contents* is stripped to allow embedding of leading spaces.

This request can't be used before the first page has started—if you get an error, simply insert `.br` before the `output` request.

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 *contents* 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 to their source equivalent: nodes such as those produced by the `\N` escape sequence will remain nodes, so the result cannot be guaranteed to be a pure string. See Section 5.24.2 [Copy Mode], page 174. Glyph parameters such as the type face and size are not preserved; use `unformat` to achieve that.

`.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 adjustable again. The vertical sizes of lines are not preserved, but glyph information (font, type size, space width, and so on) is retained.

## 5.30 Punning Names

Macros, strings, and diversions share a name space; recall Section 5.5 [Identifiers], page 83. Internally, the same mechanism is used to store them. You can thus call 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. Then 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 5.16 [Line Continuation], page 127).[74] What would have happened if we had used only one backslash at a time instead?

Interpolating a string does not hide existing macro arguments. We can also place the escaped newline outside the string interpolation instead of within the string definition. 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 5.24.1 [Parameters], page 172).

Diversions can be also called with string syntax. It is sometimes convenient to copy one-line diversions to a string.

```
.di xx
the
.ft I
interpolation system
.ft
.br
.di
.ds yy This is a test of \*(xx\c
\*(yy.
    ⇒ This is a test of the interpolation system.
```

As the previous example shows, it is possible to store formatted output in strings. The `\c` escape sequence prevents the subsequent newline from being interpreted as a break (again, see Section 5.16 [Line Continuation], page 127).

Copying multi-output line diversions produces unexpected results.

---

[74] The backslash is doubled. See Section 5.24.2 [Copy Mode], page 174.

```
.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, preventing their adjustment. 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 adjustable again.

```
.box xxx
a funny
.br
test
.br
.box
.chop xxx
.unformat xxx
This is \*[xxx].
    ⇒ This is a funny test.
```

See Section 5.36 [Gtroff Internals], page 217.

## 5.31 Environments

As discussed in Section 5.27 [Deferring Output], page 187, environments store most of the parameters that control text processing. A default environment named '`0`' exists when GNU `troff` starts up; it is modified by formatting-related requests and escape sequences.

You can create new environments and switch among them. Only one is current at any given time. Active environments are managed using a *stack*, a data structure supporting "push" and "pop" operations. The current environment is at the top of the stack. The same environment name can be pushed onto the stack multiple times, possibly interleaved with others. Popping the environment stack does not destroy the current environment; it remains accessible by name and can be made current again by pushing it at any time. Environments cannot be renamed or deleted, and can only be modified when current. To inspect the environment stack, use the `pev` request; see Section 5.37 [Debugging], page 219.

Environments store the following information.

- a partially collected line, if any
- data about the most recently output glyph and line (registers .cdp, .cht, .csk, .n, .w)
- typeface parameters (size, family, style, height and slant, inter-word and inter-sentence space sizes)
- page parameters (line length, title length, vertical spacing, line spacing, indentation, line numbering, centering, right-alignment, underlining, hyphenation parameters)
- filling enablement; adjustment enablement and mode
- tab stops; tab, leader, escape, control, no-break control, hyphenation, and margin characters
- input line traps
- stroke and fill colors

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

Enter the environment *ident*, which is created if it does not already exist, using the same parameters as for the default environment used at startup. With no argument, GNU troff switches to the previous environment.

Invoking ev with an argument puts environment *ident* onto the top of the environment stack. (If it isn't already present in the stack, this is a proper push.) Without an argument, ev pops the environment stack, making the previous environment current. It is an error to pop the environment stack with no previous environment available. The read-only string-valued register .ev contains the name of the current environment—the one at the top of the stack.

```
.ev footnote-env
.fam N
.ps 6
.vs 8
.ll -.5i
.ev

...

.ev footnote-env
\[dg] Observe the smaller text and vertical spacing.
.ev
```

We can familiarize ourselves with stack behavior by wrapping the ev request with a macro that reports the contents of the .ev register to the standard error stream.

```
.de EV
.  ev \\$1
.  tm environment is now \\n[.ev]
..
.
.EV foo
.EV bar
.EV
.EV baz
.EV
.EV
.EV
```

```
error  environment is now foo
error  environment is now bar
error  environment is now foo
error  environment is now baz
error  environment is now foo
error  environment is now 0
error  error: environment stack underflow
error  environment is now 0
```

`.evc` *environment* [Request]

Copy the contents of *environment* to the current environment.

The following environment data are not copied.

- a partially collected line, if present;
- the interruption status of the previous input line (due to use of the `\c` escape sequence);
- the count of remaining lines to center, to right-justify, or to underline (with or without underlined spaces)—these are set to zero;
- the activation status of temporary indentation;
- input line traps and their associated data;
- the activation status of line numbering (which can be reactivated with '`.nm +0`'); and
- the count of consecutive hyphenated lines (set to zero).

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

The `\n[.w]` register contains the width of the last glyph formatted in the environment.

The `\n[.cht]` register contains the height of the last glyph formatted in the environment.

The `\n[.cdp]` register contains the depth of the last glyph formatted in the 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 GNU troff should place an accent) of the last glyph formatted in the environment.

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

## 5.32 Suppressing Output

\O[*num*] [Escape sequence]
Suppress GNU troff output of glyphs and geometric primitives. The sequences \O2, \O3, \O4, and \O5 are intended for internal use by grohtml.

'\O0' Disable the emission of glyphs and geometric primitives to the output driver, provided that this sequence occurs at the outermost level (see \O3 and \O4 below). Horizontal motions corresponding to non-overstruck glyph widths still occur.

'\O1' Enable the emission of glyphs and geometric primitives to the output driver, provided that this sequence occurs at the outermost level.

\O0 and \O1 also reset the four registers opminx, opminy, opmaxx, and opmaxy to −1. These four registers mark the top left and bottom right hand corners of a box encompassing all written glyphs.

'\O2' Provided that this sequence occurs at the outermost level, enable emission of glyphs and geometric primitives, and write to the standard error stream the page number and values of the four aforementioned registers encompassing glyphs written since the last interpolation of a \O sequence, as well as the page offset, line length, image file name (if any), horizontal and vertical device motion quanta, and input file name. Numeric values are in basic units.

'\O3' Begin a nesting level. This is an internal mechanism for grohtml while producing images. At startup, gtroff is at the outermost level. These sequences are generated when processing the source document with pre-grohtml, which uses gtroff with the ps output device, Ghostscript, and the PNM tools to produce images in PNG format. They start a new page if the device is not html or xhtml, to reduce the possibility of images crossing a page boundary.

'\O4' End a nesting level.

'\O[5*Pfile*]'
Provided that this sequence occurs at the outermost level, write the name file to the standard error stream at position

*P*, which must be one of `l`, `r`, `c`, or `i`, corresponding to left, right, centered, and inline alignments within the document, respectively. *file* is a name associated with the production of the next image.

`\n[.O]` [Register]
Output suppression nesting level applied by `\O3` and `\O4` escape sequences.

## 5.33 I/O

`gtroff` has several requests for including files:

`.so` *file* [Request]
`.soquiet` *file* [Request]
Replace the `so` request's control line with the contents of the file named by the argument, "sourcing" it. *file* is sought in the directories specified by `-I` command-line option. If *file* does not exist, a warning in category '`file`' is produced and the request has no further effect. See Section 5.37.1 [Warnings], page 222, for information about the enablement and suppression of warnings.

`so` can be useful for large documents; e.g., allowing each chapter of a book to be kept in a separate file. However, files interpolated with `so` are not preprocessed; to overcome this limitation, see the *gsoelim*(1) man page.

Since GNU `troff` replaces the entire control line with the contents of a file, it matters whether `file` is terminated with a newline or not. Assume that file `xxx` contains only the word '`foo`' without a trailing newline.

```
$ printf 'foo' > xxx

The situation is
.so xxx
bar.
    ⇒ The situation is foobar.
```

`soquiet` works the same way, except that no warning diagnostic is issued if *file* does not exist.

`.pso` *command* [Request]
Read the standard output from the specified *command* and include it in place of the `pso` request.

It is an error to use this request in safer mode, which is the default. Invoke GNU `troff` or a front end with the `-U` option to enable unsafe mode.

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

`.mso` *file* [Request]
`.msoquiet` *file* [Request]
Identical to the `so` and `soquiet` requests, respectively, 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, these requests try to include `tmac.`*name* and vice versa.

`.trf` *file* [Request]
`.cf` *file* [Request]
Transparently output the contents of *file*. Each line is output as if it were preceded by `\!`; however, the lines are *not* subject to copy mode interpretation. If the file does not end with a newline, then a newline is added (`trf` only). For example, to define a macro `x` containing the contents of file `f`, use

```
.ev 1
.di x
.trf f
.di
.ev
```

The calls to `ev` prevent the partially collected output line from becoming part of the diversion (see Section 5.29 [Diversions], page 197).

Both `trf` and `cf`, when used in a diversion, embed a node (see Section 5.36 [Gtroff Internals], page 217) in it that, when reread, causes the contents of *file* to be transparently copied to the output. In AT&T `troff`, the contents of *file* are 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 5.5 [Identifiers], page 83).

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.

It is an error to use this request in safer mode, which is the default. Invoke GNU `troff` or a front end with the `-U` option to enable 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.

It is an error to use this request in safer mode; this is the default. Give GNU `troff` or a front end program the `-U` option to enable unsafe mode.

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`) write `nr` requests that set the registers `H`, `M`, and `S` to a temporary file. The `roff` document then reads the temporary file using the `so` request.

The registers `seconds`, `minutes`, and `hours`, initialized at startup of GNU `troff`, should satisfy most requirements. 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]
    ⇒ 02:17:54
```

The writable register `systat` 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.

It is an error to use these requests in safer mode; this is the default. Give GNU `troff` or a front end program the `-U` option to enable 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: an initial neutral double quote in *contents* is stripped to allow embedding of leading spaces, 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 sequence]
`\V(`*ev* [Escape sequence]
`\V[`*env*`]` [Escape sequence]
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 even in copy mode (see Section 5.24.2 [Copy Mode], page 174).

## 5.34 Postprocessor Access

Two escape sequences and two requests enable documents to pass information directly to a postprocessor. These are useful for exercising device-specific capabilities that the `groff` language does not abstract or generalize; examples include the embedding of hyperlinks and image files. Device-specific functions are documented in each output driver's man page, such as *gropdf*(1), *grops*(1), or *grotty*(1).

`.device` *xxx* ... [Request]
`\X'`*xxx* ...`'` [Escape sequence]
Embed all *xxx* arguments into GNU `troff` output as parameters to a device control command '`x X`'. The meaning and interpretation of such parameters is determined by the output driver or other postprocessor.

The `device` request processes its arguments in copy mode (see Section 5.24.2 [Copy Mode], page 174). An initial neutral double quote in *contents* is stripped to allow embedding of leading spaces. By contrast, within `\X` arguments, the escape sequences `\&`, `\)`, `\%`, and `\:` are ignored; `\SP` and `\~` are converted to single space characters; and `\\` has its escape character stripped. So that the basic Latin subset of the Unicode character set[75] can be reliably encoded in device control commands, seven special character escape sequences ('`\-`', '`\aq`', '`\dq`', '`\ga`', '`\ha`', '`\rs`', and '`\ti`',) are mapped to basic Latin glyphs; see the *groff_char*(7) man page. The use of any other escape sequence in `\X` arguments is normally an error.

If the `use_charnames_in_special` directive appears in the output device's `DESC` file, the use of special character escape sequences is *not* an error; they are simply output verbatim (with the exception of the seven mapped to Unicode basic Latin characters, discussed above). `use_charnames_in_special` is currently employed only by `grohtml`.

`.devicem` *name* [Request]
`\Y`*n* [Escape sequence]
`\Y(`*nm* [Escape sequence]
`\Y[`*name*`]` [Escape sequence]
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 AT&T `troff` output format, and confuses drivers that do not know about this extension (see Section 6.1.2.4 [Device Control Commands], page 239).

## 5.35 Miscellaneous

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

`.nm` [*start* [*inc* [*space* [*indent*]]]] [Request]
`\n[ln]` [Register]
`\n[.nm]` [Register]
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

[75] that is, ISO 646:1991-IRV or, popularly, "US-ASCII"

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 environment (see Section 5.31 [Environments], page 204).

While line numbering is enabled, the output line number register `ln` is updated as each line is output, even if no line number is formatted with it because it is being skipped (it is not a multiple of *inc*) or because numbering is suppressed (see the `nn` request below).

The `.nm` register tracks the enablement status of line numbering. Temporary suspension of numbering with the `nn` request does *not* alter its value.

```
.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]
`\n[.nn]` [Register]

Suppress numbering of the next *skip* output lines that would otherwise be numbered. The default is 1. `nn` can be invoked when line numbering is not active; suppression of numbering will take effect for *skip* lines once `nm` enables it.

The `.nn` register stores the count of output lines still to have their numbering suppressed.

This count is associated with the environment (see Section 5.31 [Environments], page 204).

To test whether the current output line will be numbered, you must check both the `.nm` and `.nn` registers.

```
.de is-numbered
.  nop This line
.  ie (\\n[.nm] & (1-\\n[.nn])) IS
.  el                           ISN'T
.  nop numbered.
.  br
..
Test line numbering.
.is-numbered
.nm 1
.nn 2
.is-numbered
.is-numbered
.is-numbered
.nm
.is-numbered
```

The output lines correctly report their numbering status.

```
Test line numbering.  This line ISN'T numbered.
This line ISN'T numbered.
This line ISN'T numbered.
  1 This line IS numbered.
This line ISN'T numbered.
```

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

Print a *margin character* to the right of the text.[76] 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; the default is 10 points. 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 environment (see Section 5.31 [Environments], page 204).

This is quite useful for indicating text that has changed, and, in fact, there are programs available for doing this (they are called `nrchbar` and `changebar` and can be found in any '`comp.sources.unix`' archive).

```
.ll 3i
.mc |
This paragraph is highlighted with a margin
character.
.sp
Vertical space isn't marked.
.br
\&
.br
But we can fake it with '\&'.
```

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

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 registers `llx`, `lly`, `urx`, and `ury`. If an error occurs (for example, `psbb` cannot find the `%%BoundingBox` comment), it sets the four registers to zero.

The search path for *filename* can be controlled with the `-I` command-line option.

## 5.36 `gtroff` Internals

`gtroff` processes input in three steps. One or more input characters are converted to an *input token*.[77] 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 5.5 [Identifiers], page 83, for more on this topic.

For example, the input string '`fi\[:u]`' is converted into a character token '`f`', a character token '`i`', and a special token '`:u`' (representing u umlaut). Later on, the character tokens '`f`' and '`i`' are merged to a single output node representing the ligature glyph '`fi`' (provided the current font has a glyph for this ligature); the same happens with '`:u`'. All output glyph nodes are 'processed', which means that they are invariably associated with a given font, font size, advance width, etc. During the formatting process, `gtroff` itself adds various nodes to control the data flow.

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

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 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 pending output 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 adjustable 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 compatibility mode enablement.

```
.cp 1     \" switch to compatibility mode
.de xx
\\$1
..
.cp 0     \" switch compatibility mode off
.xx caf\['e]
    ⇒ café
```

Since compatibility mode is enabled while `de` is invoked, the macro `xx` enables compatibility mode when it is called. Argument `$1` can still be handled properly because it inherits the compatibility mode enablement status that was active at the point where `xx` was called.

After interpolation of the parameters, the compatibility save and restore tokens are removed.

## 5.37 Debugging

> *Standard troff voodoo, just put a power of two backslashes in front of it until it works and if you still have problems add a \c.*
> — Ron Natalie

GNU `troff` is not the easiest language to debug, in part thanks to its design features of recursive interpolation and the use of multi-stage pipeline processing in the surrounding system. Nevertheless there exist several features useful for troubleshooting.

Preprocessors use the `lf` request to preserve the identity of the line numbers and names of input files. GNU `troff` emits a variety of error diagnostics and supports several categories of warning; the output of these can be selectively suppressed. Backtraces can be enabled when errors or warnings occur, or triggered on demand. The `tm` and related requests can be used to emit customized diagnostic messages or for instrumentation while troubleshooting. The `ex` and `ab` requests cause early termination with successful and error exit codes respectively, to halt further processing when continuing would be fruitless. The state of the formatter can be examined with requests that write lists of defined names (macros, strings, and diversions), environments, registers, and page location traps to the standard error stream.

`.lf` *line* [*filename*] [Request]

Change the line number and optionally the file name GNU `troff` shall use for error and warning messages. *line* is the input line number of the *next* line. Without an argument, the request is ignored.

This request is primarily a debugging aid for documents that undergo preprocessing. Programs like `tbl` that transform input in their own languages to `roff` requests use it so that any diagnostic messages emitted by `troff` correspond to the original source document.

`.tm` *contents* [Request]
`.tm1` *contents* [Request]
`.tmc` *contents* [Request]
Send *contents*, which consumes the remainder of the input line, to the standard error stream.

The `tm` request ignores leading spaces of *contents*; `tm1` handles its argument similarly to the `ds` request: an initial neutral double quote is stripped to allow embedding of leading spaces.

The `tmc` request is similar to `tm1` but does not append a newline (as is done in `tm` and `tm1`).

`.ab` [*contents*] [Request]
Write any *contents* to the standard error stream (like `tm`) and then abort GNU `troff`; that is, stop processing and terminate with a failure status.

`.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 5.33 [I/O], page 208.

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 state of the current environment followed by that of all other environments 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 registers to the standard error stream.

`.ptr` [Request]
Report the names and positions of all page location traps to the standard error stream. Empty slots in the list, where a trap has been planted but subsequently (re)moved, are printed as well.

`.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]
Write a backtrace of the input stack to the standard error stream.

Consider the following in a file `test`.

```
.de xxx
.  backtrace
..
.de yyy
.  xxx
..
.
.yyy
```

```
error  troff: backtrace: 'test':2: macro 'xxx'
error  troff: backtrace: 'test':5: macro 'yyy'
error  troff: backtrace: file 'test':8
```

The `-b` option of GNU `troff` causes a backtrace to be generated on each error or warning. Some warnings have to be enabled; See Section 5.37.1 [Warnings], page 222.

`\n[slimit]` [Register]
If greater than 0, sets the maximum quantity of objects on GNU `troff`'s internal input stack. If less than or equal to 0, there is no limit: recursion can continue until program memory is exhausted. The default is 1,000.

`.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 inter-word space in a line. See Section 5.37.1 [Warnings], page 222.

GNU `troff` has command-line options for reporting warnings (`-w`) and backtraces (`-b`) when a warning or an error occurs.

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

Select the categories, or "types", of reported warnings. *n* is the sum of the numeric codes associated with each warning category that is to be enabled; all other categories are disabled. The categories and their associated codes are listed in Section 5.37.1 [Warnings], page 222. For example, '`.warn 0`' disables all warnings, and '`.warn 1`' disables all warnings except those about missing glyphs. If no argument is given, all warning categories are enabled.

The read-only register `.warn` contains the sum of the numeric codes of enabled warning categories.

## 5.37.1 Warnings

Warning diagnostics emitted by GNU `troff` are divided into named, numbered categories. The name associated with each warning category is used by the `-w` and `-W` options. Each category is also assigned a power of two; the sum of enabled category values is used by the `warn` request and the `.warn` register.

Warnings of each category are produced under the following circumstances.

'`char`'
'`1`' An undefined glyph was requested for output.[78] This category is enabled by default.

'`number`'
'`2`' An invalid numeric expression was encountered. This category is enabled by default. See Section 5.4 [Numeric Expressions], page 79.

'`break`'
'`4`' A filled output line could not be broken such that its length was less than the output line length '`\n[.l]`'. This category is enabled by default.

[78] `char` is a misnomer since it reports missing glyphs—there are no "missing" input characters, only invalid ones.

'`delim`'
'`8`' The closing delimiter in an escape sequence was missing or mismatched.

'`el`'
'`16`' The `el` request was encountered with no prior corresponding `ie` request. See Section 5.23.3 [if-else], page 164.

'`scale`'
'`32`' A scaling unit inappropriate to its context was used in a numeric expression.

'`range`'
'`64`' A numeric expression was out of range for its context.

'`syntax`'
'`128`' A self-contradictory hyphenation mode was requested; an empty or incomplete numeric expression was encountered; an operand to a numeric operator was missing; an attempt was made to define a recursive, empty, or nonsensical character class; or a `groff` extension conditional expression operator was used while in compatibility mode.

'`di`'
'`256`' A `di`, `da`, `box`, or `boxa` request was invoked without an argument when there was no current diversion.

'`mac`'
'`512`' An undefined string, macro, or diversion was used. When such an object is dereferenced, an empty one of that name is automatically created. So, unless it is later deleted, at most one warning is given for each.

This warning is also emitted upon an attempt to move an unplanted trap macro (see Section 5.28.1.1 [Page Location Traps], page 188). In such cases, the unplanted macro is *not* dereferenced, so it is not created if it does not exist.

'`reg`'
'`1024`' An undefined register was used. When an undefined register is dereferenced, it is automatically defined with a value of 0. So, unless it is later deleted, at most one warning is given for each.

'`tab`'
'`2048`' A tab character was encountered where a number was expected, or appeared in an unquoted macro argument.

'`right-brace`'
'`4096`' A right brace escape sequence `\}` was encountered where a number was expected.

'`missing`'
'`8192`' A request was invoked with a mandatory argument absent.

‘input’
‘16384’ An invalid character occurred on the input stream.

‘escape’
‘32768’ An unsupported escape sequence was encountered.

‘space’
‘65536’ A space was missing between a request or macro and its argument. This warning is produced when an undefined name longer than two characters is encountered and the first two characters of the name constitute a defined name. No request is invoked, no macro called, and an empty macro is not defined. This category is enabled by default. It never occurs in compatibility mode.

‘font’
‘131072’ A non-existent font was selected, or the selection was ignored because a font selection escape sequence was used after the output line continuation escape sequence on an input line. This category is enabled by default.

‘ig’
‘262144’ An invalid escape sequence occurred in input ignored using the `ig` request. This warning category diagnoses a condition that is an error when it occurs in non-ignored input.

‘color’
‘524288’ An undefined color was selected, an attempt was made to define a color using an unrecognized color space, an invalid component in a color definition was encountered, or an attempt was made to redefine a default color.

‘file’
‘1048576’ An attempt was made to load a file that does not exist. This category is enabled by default.

Two warning names group other warning categories for convenience.

‘all’ All warning categories except ‘`di`’, ‘`mac`’ and ‘`reg`’. This shorthand is intended to produce all warnings that are useful with macro packages written for AT&T `troff` and its descendants, which have less fastidious diagnostics than GNU `troff`.

‘w’ All warning categories. Authors of documents and macro packages targeting `groff` are encouraged to use this setting.

## 5.38 Implementation Differences

GNU `troff` has a number of features that cause incompatibilities with documents written for other versions of `troff`. Some GNU extensions to `troff` have become supported by other implementations.

### 5.38.1 Safer Mode

The formatter operates in "safer" mode by default; to mitigate risks from untrusted input documents, the `pi` and `sy` requests are disabled. GNU `troff`'s `-U` option enables "unsafe mode", restoring their function and enabling additional `groff` extension requests, `open`, `opena`, and `pso`. See Section 5.33 [I/O], page 208.

### 5.38.2 Compatibility Mode

Long identifier 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 an interpolation of a string or register, respectively, named '`[`'. In GNU `troff`, however, the '`[`' is normally interpreted as delimiting a long name. In compatibility mode, GNU `troff` interprets names in the traditional way; they thus can be two characters long at most.

`.cp` [*n*] [Request]
`\n[.C]` [Register]

If *n* is missing or non-zero, turn on compatibility mode; otherwise, turn it off.

The read-only register `.C` is 1 if compatibility mode is on, 0 otherwise.

Compatibility mode can be also turned on with the `-C` command-line option.

`.do` *name* [Request]
`\n[.cp]` [Register]

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, or diversion *name* as well as data read from files or pipes if *name* is any of the `so`, `soquiet`, `mso`, `msoquiet`, or `pso` requests.

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 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 enablement in the surrounding context. It may occur to you to save the existing value of '`\n(.C`' into a register, say, '`_C`', at the beginning of your file, turn compatibility mode off with '`.cp 0`', then restore it from that register at the end with '`.cp \n(_C`'. At the same time, a modular design of a document or macro package may lead you to multiple layers of inclusion. You cannot use the same register name everywhere lest you "clobber" the value from a preceding or enclosing context. The two-character register name space of AT&T `troff` is confining 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.

To robustly save compatibility mode before switching it off, use

```
.do nr _my_saved_C \n[.cp]
.cp 0
```

at the beginning of your file, followed by

```
.cp \n[_my_saved_C]
.do rr _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 interpolation depth 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 escape sequences \f, \H, \m, \M, \R, \s, and \S are transparent for the purpose of recognizing a control character at 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)

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[79] should be migrated to another \s form.

### 5.38.3 Other Differences

groff request names unrecognized by other troff implementations will likely be ignored by them; escape sequences that are groff extensions are liable to be interpreted as if the escape character were not present. For example, the adjustable, non-breaking escape sequence \~ is also supported by Heirloom Doctools troff 050915 (September 2005), mandoc 1.9.5 (2009-09-21), neatroff (commit 1c6ab0f6e, 2016-09-13), and Plan 9 from User Space troff (commit 93f8143600, 2022-08-12), but not by Solaris or Documenter's

[79] The Graphic Systems C/A/T phototypesetter (the original device target for AT&T troff) supported only a few discrete type sizes in the range 6–36 points, so Ossanna contrived a special case in the parser to do what the user must have meant. Kernighan warned of this in the 1992 revision of CSTR #54 (§2.3), and more recently, McIlroy referred to it as a "living fossil".

Workbench troffs. See Section 5.9 [Manipulating Filling and Adjustment], page 101.

GNU troff does not allow the use of the escape sequences \|, \^, \&, \{, \}, \SP, \', \`, \-, _, \!, \%, and \c in identifiers; AT&T troff does. The \A escape sequence (see Section 5.5 [Identifiers], page 83) may be helpful in avoiding use of these escape sequences in names.

When adjusting to both margins, AT&T troff at first adjusts spaces starting from the right; GNU troff begins from the left. Both implementations adjust spaces from opposite ends on alternating output lines in this adjustment mode to prevent "rivers" in the text.

GNU troff does not always hyphenate words as AT&T troff does. The AT&T implementation uses a set of hard-coded rules specific to 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.

GNU troff predefines a string .T containing the argument given to the -T command-line option, namely the current output device (for example, 'pdf' or 'utf8'). The existence of this string is a common feature of post-CSTR #54 troffs[80] but valid values are specific to each implementation.

The (read-only) register .T interpolates 1 if GNU troff is called with the -T command-line option, and 0 otherwise. This behavior differs from AT&T troff, which interpolated 1 only if nroff was the formatter and was called with -T.

AT&T troff and other implementations handle the lf request differently. For them, its *line* argument changes the line number of the *current* line.

AT&T troff had only environments named '0', '1', and '2'. In GNU troff, any number of environments may exist, using any valid identifiers for their names (see Section 5.5 [Identifiers], page 83.)

Fractional type sizes cause one noteworthy incompatibility. In AT&T troff the ps request ignores scale indicators and thus '.ps 10u' sets the type size to 10 points, whereas in GNU troff it sets the type size to 10 *scaled* points. See Section 5.20.3 [Using Fractional Type Sizes], page 154.

The ab request differs from AT&T troff: GNU troff writes no message to the standard error stream if no arguments are given, and it exits with a failure status instead of a successful one.

The bp request differs from AT&T troff: GNU troff does not accept a scaling indicator on the argument, a page number; the former (somewhat uselessly) does.

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.

[80] DWB 3.3, Solaris, Heirloom Doctools, and Plan 9 troff all support it.

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 space 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; afterward, 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,[81] 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 or \[rs], for "reverse solidus", from its name in the ECMA-6 (ISO/IEC 646) standard.[82]

To store an escape sequence in a diversion that is interpreted when the diversion is reread, either use the traditional \! transparent output facil-

[81] 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.

[82] The rs special character identifier was not defined in AT&T troff's font description files, but is in those if its lineal descendant, Heirloom Doctools troff, as of the latter's 060716 release (July 2006).

ity, or, if this is unsuitable, the new `\?` escape sequence. See Section 5.29 [Diversions], page 197, and Section 5.36 [Gtroff Internals], page 217.

In the somewhat pathological case where a diversion exists containing a partially collected line and a partially collected line at the top-level diversion has never existed, AT&T `troff` will output the partially collected line at the end of input; GNU `troff` will not.

# 6 File Formats

All files read and written by `gtroff` are text files. The following two sections describe their format.

## 6.1 `gtroff` Output

This section describes the `groff` intermediate output format produced by GNU `troff`.

As `groff` is a wrapper program around GNU `troff` and automatically calls an output driver (or "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 as it is when calling GNU `troff` directly.

Here, the term *troff output* describes what is output by GNU `troff`, while *intermediate output* refers to the language that is accepted by the parser that prepares this output for the output drivers. This parser handles whitespace more flexibly than AT&T's implementation 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`.

### 6.1.1 Language Concepts

The fundamental operation of the GNU `troff` formatter is the translation of the `groff` input language into a device-independent form primarily concerned with what has to be written or drawn at specific positions on the output device. This language is simple and imperative. In the following discussion, the term *command* always refers to this intermediate output language, and never to the `groff` language intended for direct use by document authors. Intermediate output commands comprise several categories: glyph output; font, color, and text size selection; motion of the printing position; page advancement; drawing of geometric primitives; and device control commands, a catch-all for operations not easily classified as any of the foregoing, such as directives to start and stop output, identify the intended output device, or place URL hyperlinks in supported output formats.

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

#### 6.1.1.1 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 span several input 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.

#### 6.1.1.2 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 scaling 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.

#### 6.1.1.3 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 6.1.2.4 [Device Control Commands], page 239. The parser for the intermediate output format is able to interpret 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.

### 6.1.2 Command Reference

This section describes all intermediate output commands, both from AT&T `troff` as well as the `gtroff` extensions.

#### 6.1.2.1 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.

#### 6.1.2.2 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 AT&T `troff` manual allows negative values for *n* also, but GNU `troff` doesn't use them.

`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 65535. 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 65535 (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 output. 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 output 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 type size to *n* scaled points (this is unit '`z`'). AT&T `troff` used the unit points ('`p`') instead. See Section 6.1.4 [Output Language Compatibility], page 243.

`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 type size, and rounded to a multiple of the horizontal motion quantum. 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 6.2.1 [DESC File Format], page 244).

`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 6.2.1 [DESC File Format], page 244).

`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 AT&T `troff` manual allows negative values for *n* also, but GNU `troff` doesn't use them.

`w` Describe an adjustable space. This performs no action; it is present for documentary purposes. The spacing itself must be performed explicitly by a move command.

### 6.1.2.3 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 5.26 [Drawing Requests], page 182.

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 65535. 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 65535 (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 65535
    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 type 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.

### 6.1.2.4 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 6.1.1.1 [Separation], page 232).

`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 5.19.3 [Font Positions], page 135.

`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 6.1.4 [Output Language Compatibility], page 243.

`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 AT&T `troff` manual documents this command as

  > pause device, can be restarted

  but GNU `troff` output drivers do nothing with this command.

`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 the name of the output driver to *xxx*, a sequence of non-whitespace characters terminated by whitespace. The possible names correspond to those of `groff`'s `-T` option. This is the first command of the 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.

#### 6.1.2.5 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.

### 6.1.3 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.

### 6.1.4 Output Language Compatibility

The intermediate output language of AT&T `troff` was first documented in *A Typesetter-independent TROFF*, by Brian Kernighan, and by 1992 the AT&T `troff` manual was updated to incorprate a description of it.

The GNU `troff` 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.

## 6.2 Device and Font Description Files

The `groff` font and output device description formats are slight extensions of those used by AT&T device-independent `troff`. In distinction to the AT&T implementation, `groff` lacks a binary format; all files are text files.[3] The device and font description files for a device *name* are stored in a `dev`*name* directory. The device description file is called `DESC`, and, for each font supported by the device, a font description file is called *f*, where *f* is usually an abbreviation of a font's name and/or style. For example, the `ps` (PostScript) device has `groff` font description files for Times roman (`TR`) and Zapf Chancery Medium italic (`ZCMI`), among many others, while the `utf8` device (for terminal emulators) has only font descriptions for the roman, italic, bold, and bold-italic styles (`R`, `I`, `B`, and `BI`, respectively).

Device and font description files are read both by the formatter, GNU `troff`, and by output drivers. The programs delegate these files' processing to an internal library, `libgroff`, ensuring their consistent interpretation.

### 6.2.1 `DESC` File Format

The `DESC` file contains a series of directives; each begins a line. Their order is not important, with two exceptions: (1) the `res` directive must precede any `papersize` directive; and (2) the `charset` directive must come last (if at all). If a directive name is repeated, later entries in the file override previous ones (except that the paper dimensions are computed based on the `res` directive last seen when `papersize` is encountered). Spaces and/or tabs

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

separate words and are ignored at line boundaries. Comments start with the '`#`' character and extend to the end of a line. Empty lines are ignored.

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

`fonts` *n F1 ... Fn*
: Fonts *F1*, ..., *Fn* are mounted at font positions *m*+1, ..., *m*+*n* where *m* is the number of `styles` (see below). This directive may extend over more than one line. A font name of `0` causes no font to be mounted at the corresponding position.

`hor` *n*
: The horizontal motion quantum is *n* basic units. All horizontal quantities are rounded to multiples of *n*.

`image_generator` *program*
: Use *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. The `grohtml` driver uses this directive.

`paperlength` *n*
: The vertical dimension of the output medium is *n* basic units (deprecated: use `papersize` instead).

`papersize` *format-or-dimension-pair-or-file-name ...*
: The dimensions of the output medium are as according to the argument, which is either a standard paper format, a pair of dimensions, or the name of a plain text file containing either of the foregoing.

  Recognized paper formats are the ISO and DIN formats `A0`–`A7`, `B0`–`B7`, `C0`–`C7`, `D0`–`D7`; the U.S. paper types `letter`, `legal`, `tabloid`, `ledger`, `statement`, and `executive`; and the envelope formats `com10`, `monarch`, and `DL`. Matching is performed without regard for lettercase.

  Alternatively, the argument can be a custom paper format in the format *`length,width`* (with no spaces before or after the comma). Both *length* and *width* must have a unit appended; valid units are '`i`' for inches, '`c`' for centimeters, '`p`' for points, and '`P`' for picas. Example: '`12c,235p`'. An argument that starts with a digit is always treated as a custom paper format.

  Finally, the argument can be a file name (e.g., `/etc/papersize`); if the file can be opened, the first line is read and a match attempted against each of the other forms. No comment syntax is supported.

  More than one argument can be specified; each is scanned in turn and the first valid paper specification used.

`paperwidth` *n*
: The horizontal dimension of the output medium is *n* basic units (deprecated: use `papersize` instead).

`pass_filenames`
: Direct GNU `troff` to emit the name of the source file being processed. This is achieved with the intermediate output command '`x F`', which `grohtml` interprets.

`postpro` *program*
: Use *program* as the postprocessor.

`prepro` *program*
: Use *program* as a preprocessor. The `html` and `xhtml` output devices use this directive.

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

`res` *n*
: The device resolution is *n* basic units per inch.

`sizes` *s1* ... *sn* `0`
: The device has fonts at *s1*, ..., *sn* scaled points (see below). The list of sizes must be terminated by `0`. Each *si* can also be a range of sizes *m*–*n*. The list can extend over more than one line.

`sizescale` *n*
: A typographical point is subdivided into *n* scaled points. The default is `1`. See Section 5.20.3 [Using Fractional Type Sizes], page 154.

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

`tcommand`
: The postprocessor can handle the '`t`' and '`u`' intermediate output commands.

`unicode`
: The output device supports the complete Unicode repertoire. This directive is useful only for devices 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 font description file's `charset` section, they either override the default mappings for those particular characters or add new mappings (normally for composite characters).

  The `utf8`, `html`, and `xhtml` output devices use this directive.

`unitwidth` *n*
: Quantities in the font description files are in basic units for fonts whose type size is *n* scaled points.

`unscaled_charwidths`
: Make the font handling module always return unscaled character widths. The `grohtml` driver uses this directive.

`use_charnames_in_special`
: GNU `troff` should encode special characters inside device control commands; see Section 5.34 [Postprocessor Access], page 212. The `grohtml` driver uses this directive.

`vert` *n*
: The vertical motion quantum is *n* basic units. All vertical quantities are rounded to multiples of *n*.

`charset`
: This line and everything following it in the file are ignored. It is recognized for compatibility with other `troff` implementations. In GNU `troff`, character set repertoire is described on a per-font basis.

GNU `troff` recognizes but ignores the directives `spare1`, `spare2`, and `biggestfont`.

The `res`, `unitwidth`, `fonts`, and `sizes` lines are mandatory. Directives not listed above are ignored by GNU `troff` but may be used by postprocessors to obtain further information about the device.

### 6.2.2 Font Description File Format

On typesetting output devices, each font is typically available at multiple sizes. While paper measurements in the device description file are in absolute units, measurements applicable to fonts must be proportional to the type size. `groff` achieves this using the precedent set by AT&T device-independent `troff`: one font size is chosen as a norm, and all others are scaled linearly relative to that basis. The "unit width" is the number of basic units per point when the font is rendered at this nominal size.

For instance, `groff`'s `lbp` device uses a `unitwidth` of 800. Its Times roman font '`TR`' has a `spacewidth` of 833; this is also the width of its comma, period, centered period, and mathematical asterisk, while its '`M`' is 2,963 basic units. Thus, an '`M`' on the `lbp` device is 2,963 basic units wide at a notional type size of 800 points.[4]

A font description file has two sections. The first is a sequence of directives, and is parsed similarly to the `DESC` file described above. Except for the directive names that begin the second section, their ordering is immaterial. Later directives of the same name override earlier ones, spaces and tabs are handled in the same way, and the same comment syntax is supported. Empty lines are ignored throughout.

[4] 800-point type is not practical for most purposes, but using it enables the quantities in the font description files to be expressed as integers.

`name` *f*
: The name of the font is *f*. '`DESC`' is an invalid font name. Simple integers are valid, but their use is discouraged.[5]

`spacewidth` *n*
: The width of an unadjusted inter-word space is *n* basic units.

The directives above must appear in the first section; those below are optional.

`slant` *n*
: The font's glyphs have a slant of *n* degrees; a positive *n* slants in the direction of text flow.

`ligatures` *lig1* ... *lign* [`0`]
: Glyphs *lig1*, ..., *lign* are ligatures; possible ligatures are '`ff`', '`fi`', '`fl`', '`ffi`' and '`ffl`'. For compatibility with other `troff` implementations, the list of ligatures may be terminated with a `0`. The list of ligatures must not extend over more than one line.

`special`
: The font is *special*: when a glyph is requested that is not present in the current font, it is sought in any mounted fonts that bear this property.

Other directives in this section are ignored by GNU `troff`, but may be used by postprocessors to obtain further information about the font.

The second section contains one or two subsections. These can appear in either order; the first one encountered commences the second section. Each starts with a directive on a line by itself. A `charset` subsection is mandatory unless the associated `DESC` file contains the `unicode` directive. Another subsection, `kernpairs`, is optional.

The directive `charset` starts the character set subsection.[6] It precedes a series of glyph descriptions, one per line. Each such glyph description comprises a set of fields separated by spaces or tabs and organized as follows.

*name metrics type code* [*entity-name*] [`--` *comment*]

*name* identifies the glyph: if *name* is a printable character *c*, it corresponds to the `troff` ordinary character *c*. If *name* is a multi-character sequence not beginning with `\`, it corresponds to the GNU `troff` special character escape sequence '`\[`*name*`]`'. A name consisting of three minus signs, '`---`', is special and indicates that the glyph is unnamed: such glyphs can be accessed only by the `\N` escape sequence in `troff`. A special character named '`---`' can still be defined using `char` and similar requests. The *name* '`\-`' defines the minus sign glyph. Finally, *name* can be the unbreakable one-sixth and one-twelfth space escape sequences, `\|` and `\^` ("thin" and "hair" spaces, respectively),

[5] `groff` requests and escape sequences interpret non-negative font names as mounting positions instead. Further, a font named '`0`' cannot be automatically mounted by the `fonts` directive of a `DESC` file.

[6] For typesetter devices, this directive is misnamed since it starts a list of glyphs, not characters.

in which case only the width metric described below is interpreted; a font can thus customize the widths of these spaces.

The form of the *metrics* field is as follows.

*width*[**,**[*height*[**,**[*depth*[**,**[*italic-correction*
[**,**[*left-italic-correction*[**,**[*subscript-correction*]]]]]]]]]

There must not be any spaces, tabs, or newlines between these *subfields* (which have been split here into two lines only for better legibility). The subfields are in basic units expressed as decimal integers. Unspecified subfields default to `0`. Since there is no associated binary format, these values are not required to fit into the C language data type '`char`' as they are in AT&T device-independent `troff`.

The *width* subfield gives the width of the glyph. The *height* subfield gives the height of the glyph (upward is positive); if a glyph does not extend above the baseline, 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 below the baseline to which the glyph extends (downward is positive); if a glyph does not extend below the baseline, it should be given a zero depth, rather than a negative depth. Italic corrections are relevant to glyphs in italic or oblique styles. The *italic-correction* is the amount of space that should be added after an oblique glyph to be followed immediately by an upright glyph. The *left-italic-correction* is the amount of space that should be added before an oblique glyph to be preceded immediately by an upright glyph. The *subscript-correction* is the amount of space that should be added after an oblique glyph to be followed by a subscript; it should be less than the italic correction.

For fonts used with typesetting devices, the *type* field gives a featural description of the glyph: it is a bit mask recording whether the glyph is an ascender, descender, both, or neither. When a `\w` escape sequence is interpolated, these values are bitwise or-ed together for each glyph and stored in the `nr` register. In font descriptions for terminal devices, all glyphs might have a type of zero, regardless of their appearance.

`0` means the glyph lies entirely between the baseline and a horizontal line at the "x-height" of the font; typical examples are '`a`', '`c`', and '`x`';

`1` means the glyph descends below the baseline, like '`p`';

`2` means the glyph ascends above the font's x-height, like '`A`' or '`b`'; and

`3` means the glyph is both an ascender and a descender—this is true of parentheses in some fonts.

The *code* field gives a numeric identifier that the postprocessor uses to render the glyph. The glyph can be specified to `troff` using this code by means of the `\N` escape sequence. *code* can be any integer.[7]

[7] that is, any integer parsable by the C standard library's `strotol` function

The *entity-name* field defines an identifier for the glyph that the postprocessor uses to print the GNU `troff` glyph *name*. This field is optional; it was introduced so that the `grohtml` output driver could encode its character set. For example, the glyph '`\[Po]`' is represented by '`&pound;`' in HTML 4.0. For efficiency, these data are now compiled directly into `grohtml`. `grops` uses the field to build sub-encoding arrays for PostScript fonts containing more than 256 glyphs. Anything on the line after the *entity-name* field or '`--`' is ignored.

A line in the `charset` section can also have the form

```
name "
```

identifying *name* as another name for the glyph mentioned in the preceding line. Such aliases can be chained.

The directive `kernpairs` starts a list of kerning adjustments to be made to adjacent glyph pairs from this font. It contains a sequence of lines formatted as follows.

```
g1 g2 n
```

The foregoing means that when glyph *g1* is typeset immediately before *g2*, the space between them should be increased by *n*. Most kerning pairs should have a negative value for *n*.

# Appendix 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.

# Appendix B Request Index

Request names appear without a leading control character; the defaults are `.` for the regular control character and `'` for the no-break control character.

## A



## B



## C



## D



## E



## F



## G

## H



## I



## K



## L



## M



## N



## O



## P



## R

## S



## T



## U



## V



## W

# Appendix C Escape Sequence Index

The escape character, \ by default, is always followed by at least one more input character, making an escape *sequence*. Any input token \X with X not in the list below emits a warning and interpolates glyph X. Note the entries for \., which may be obscured by the leader dots, and for *newline* and *space*, which are sorted alphabetically, not by code point order.

# Appendix D Operator Index

# Appendix 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



## T



## U



## V



## Y

# Appendix 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

# Appendix G String Index

The macro package or program a that defines or uses each string is appended in brackets. (Only one string, `.T`, is defined by the `troff` formatter itself.) See Section 5.22 [Strings], page 157.

## C



## D



## F



## L



## M



## O



## Q



## R



## S



## T



## U



## V

# Appendix H File Keyword Index

# Appendix I Program and File Index

# Appendix J 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



# Y



# Z