groff The GNU implementation of `troff`
Edition 1.23.0
Autumn 2020

by Trent A. Fisher

and Werner Lemberg

This manual documents GNU `troff` version 1.23.0.

6 March 2021

Table of Contents

1. Introduction

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

1.1. What Is groff?

groff belongs to an older generation of document preparation systems, which operate more like compilers than the more recent interactive WYSIWYG[1] systems. groff and its contemporary counterpart, TEX, both work using a *batch* par adigm: The input (or *source*) files are normal text files with embedded formatting commands. These files can then be processed by groff to produce a typeset document on a variety of devices.

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

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

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

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

1.2. History

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

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

When they needed a more flexible language, a new version of roff called nroff (after "new roff", pronounced "en-roff") was written. It had a much more complicated syntax, but provided the basis for all future versions. When they got a Graphic Systems CAT

[1] What You See Is What You Get

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

Phototypesetter, Ossanna wrote a version of `nroff` that would drive it. It was dubbed `troff`, for "typesetter `roff`", although many people have speculated that it actually means "Times `roff`" because of the use of the Times font family in `troff` by default. As such, the name `troff` is pronounced "tee-roff" rather than "trough".

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

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

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

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

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

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

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

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

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

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

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

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

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

1.3. groff Capabilities

So what exactly is groff capable of doing? groff pro vides 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* f acility to specify how certain routine operations (e.g. starting paragraphs, printing headers and footers, etc.) should be done. These macros can be collected together into a*macro pac kage*. There are a number of macro packages available; the most common (and the ones described in this manual) are man, mdoc, me, ms, and mm.

1.5. Preprocessors

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

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

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

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

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

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

1.6. Output Devices

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

1.7. Credits

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

Larry Kollar contributed the section on the ms macro package.

2. Invoking groff

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

It has become a tradition that GNU programs get the prefix 'g' to distinguish them from their original counterparts provided by the host (see Environment). 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 GNUtroff program. All other implementations oftroff are called A T&T troff, which is the common origin of almost all troff implementations[4] (with more or less compatible changes). Similarly, we say 'gpic', 'geqn', and so on.

2.1. Options

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

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

The command-line format for groff is:

```
groff [ -abceghijklpstvzCEGNRSUVXZ ] [ -dcs ] [ -Darg ]
      [ -ffam ] [ -Fdir ] [ -Idir ] [ -Karg ]
      [ -Larg ] [ -mname ] [ -Mdir ] [ -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.

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.

[4] Besides groff, neatroff is an exception.

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

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

'`-a`' Generate a plain text approximation of the typeset output. The read-only register `.A` is set to 1. See Built-in Registers. 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. Inter-sentence 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`' Print a backtrace with each warning or error message. This backtrace should help track down the cause of the error. The line numbers given in the backtrace may not always be correct: `gtroff` can get confused by `as` or `am` requests while counting line numbers.

'`-c`' Suppress color output.

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

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

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

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

'`-E`' Inhibit all error messages.

'`-ffam`' Use *fam* as the default font family. See Font Families.

'`-Fdir`' Search *dir* for subdirectories `devname` (*name* is the name of the device), for the `DESC` file, and for font files before looking in the standard directories (see Font Directories). This option is passed to all pre- and postprocessors using the `GROFF_FONT_PATH` environment variable.

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

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

'-h' Print a help message.

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

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

- gsoelim (see gsoelim for more details); it also implies groff's -s option.
- gtroff; it is used to search files named in the '\X'ps: import' and '\X'ps: file' escapes.

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

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

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

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

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

'-L*arg*' Pass *arg* to the spooler. Each argument should be passed with a separate -L option. groff does not prepend a '-' to*arg* bef ore passing it to the postprocessor. If theprint k eyword in the device description file is missing, -L is ignored.

'-m*name*' Read in the file *name*.tmac. Normallygroff searches f or this in its macro directories. If it isn't found, it tries tmac.*name* (searching in the same directories).

'-M*dir*' Search directory *dir* for macro files before the standard directories (see Macro Directories).

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

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

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

Within gtroff, this information can be extracted with the '.P' register. See Built-in Registers.

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

'-p' Preprocess with gpic.

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

'-rcn'
'-rname=n'
Set register *c* or *name* to the value *n*. *c* must be a one-letter name;*name* can be of arbitrary length. *n* can be an y gtroff numeric expression. All register assignments happen before loading any macro file (including the start-up file).

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

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

'-s' Preprocess with gsoelim.

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

'-t' Preprocess with gtbl.

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

ps For PostScript printers and previewers.

pdf For PDF viewers or printers.

dvi For TEX DVI format.

X75 For a 75 dpi X11 previewer.

X75-12 For a 75 dpi X11 previewer with a 12 pt base font in the document.

X100 For a 100 dpi X11 previewer.

X100-12 For a 100 dpi X11 previewer with a 12 pt base font in the document.

ascii For typewriter-like devices using the (7-bit) ASCII (ISO 646) character set.

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

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

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

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

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

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

The predefined `gtroff` string `.T` contains the current 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 `gtroff`). See Built-in Registers.

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

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

'`-w`*name*' Enable warning *name*. Available warnings are described in Debugging. Multiple `-w` options are allowed.

'`-W`*name*' Inhibit warning *name*. Multiple`-W` options are allo wed.

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

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

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

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

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

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

2.2. Environment

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

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

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

The default command prefix is determined during the installation process. If a

non-GNU troff system is found, prefix 'g' is used, none otherwise.

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

GROFF_FONT_PATH
: A colon-separated list of directories in which to search for the dev*name* directory (before the default directories are tried). See Font Directories.

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

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

GROFF_TYPESETTER
: The default output device.

SOURCE_DATE_EPOCH
: A timestamp (expressed as seconds since the Unix epoch) to use as the creation timestamp in place of the current time. The time is converted to human-readable form using*ctime*(3) when the f ormatter starts up and stored in registers usable by documents and macro packages (see Built-in Registers).

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. Themso request doesn't ha ve 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 Font Families), fonts within a family should start with the family name, followed by the shape. For example, the Times family uses '`T`' for the family name and '`R`', '`B`', '`I`', and '`BI`' to indicate the shapes 'roman', 'bold', 'italic', and 'bold italic', respectively. Thus the final font names are '`TR`', '`TB`', '`TI`', and '`TBI`'.

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

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

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

  ```
  /usr/local/share/groff/site-font
  /usr/local/share/groff/1.23.0/font
  ```

  assuming that the version of `groff` is 1.23.0, and the installation prefix was `/usr/local`. It is possible to fine-tune those directories during the installation process.

2.5. Paper Size

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

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

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

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

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

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

2.6. Invocation Examples

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

```
groff file
```

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

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

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

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

Preview `file` with `gxditview`, using the `me` macro package. Since no `-T` option is specified, use the default device ('`ps`'). You can say either '`-m me`' or '`-me`'; the latter is an anachronism from the early days of Unix.[5]

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

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

2.6.1. grog

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

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

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

For example,

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

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

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

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

3. Tutorial for Macro Users

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

3.1. Basics

This section covers some of the basic concepts necessary to understand how to use a macro package.[6] References are made throughout to more detailed information, if desired.

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

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

```
.sp
```

spaces one line, but

```
.sp 4
```

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

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

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

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

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

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

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

---

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

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`'.[7]

A number of requests allow you to change the way the output is arranged on the page, sometimes called the*la yout* 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 (f or *N* centimeters). For example, the input:

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

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

Text lines can be centered by using the `ce` request. The line after `ce` is centered (horizontally) on the page. To center more than one line, use '`.ce` *N*' (where*N* is the n umber 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 z ero 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

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

`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 pac kage*.

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

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

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*inde x*, none actually provide that functionality. The facilities they call indices are actually more appropriate for tables of contents.

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

3.2.9. Paper Formats

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

3.2.10. Multiple Columns

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

3.2.11. Font and Size Changes

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

3.2.12. Predefined Strings

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

3.2.13. Preprocessor Support

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

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

3.2.14. Configuration and Customization

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

4. Macro Packages

This chapter documents the major macro packages that come with `groff`. Such packages are also sometimes described as*full-service* due to the breadth of f eatures 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
```

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

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

  Use the `FT` 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*
: Print '<CTRL/*key*>'.

`.CW`
: Print subsequent text using a "constant-width" (monospaced) typeface (Courier roman).

`.Ds`
Begin a non-filled display.

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

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

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

`.G [`*text*`]`
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*`]`
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*`]`
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*`]`
Identical to `HB`.

`.MS` *title sect* `[`*punct*`]`
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*]
Begin a note. Print the optional*title*, or the w ord “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`
End a note begun with `NT`.

`.PN` *path* `[`*punct*`]`
Set the path name in a monospaced typeface (Courier roman), followed by optional punctuation.

`.Pn [`*punct*`]` *path* `[`*punct*`]`
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`
Switch to roman font and turn off any underlining in effect.

`.RN`
Print the string ‘`<RETURN>`’.

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

`.VE`
End printing the change bar begun by `VS`.

Simple example

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

```
.\" Make the heading fonts Helvetica
.ds HF HB
.
.\" Put more 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*g roff_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

NAME

groff_mm – memorandum macros for GNU roff

SYNOPSIS

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

DESCRIPTION

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

- No Bell Labs localisms are implemented.
- The macros OK and PM are not implemented.
- groff mm does not support cut marks. **mm** is intended to support easy localization. Use**mmse** as an e xample how to adapt the output format to a national standard. Localized strings are collected in the file

'/usr/share/groff/1.22.3.rc1.24-ea225/tmac/*xx*.tmac', where *xx* denotes the two-letter code for the *language*, as defined in the ISO 639 standard. For Swedish, this is 'sv.tmac' – not 'se', which is the ISO 3166 two-letter code for the *country* (as used for the output format localization).

A file called **locale** or *country*_**locale** is read after the initialization of the global variables. It is therefore possible to localize the macros with a different company name and so on. In this manual, square brackets are used to show optional arguments.

Number registers and strings

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

.nr *XXX* [±]*n* [*i*]\/

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

Strings are defined with **ds**.

.ds *YYY string*

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

Special formatting of number registers

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

.af *R c*

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

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

Fonts

In **mm**, the fonts (or rather, font styles) **R** (normal), **I** (italic), and **B** (bold) are hardwired to font positions **1**, **2**, and **3**, respectively. Internally, font positions are used for backwards compatibility. From a practical point of view it doesn't make a big difference – a different font family can still be selected with a call to the **.fam** request or using **groff**'s **–f** command-line option. On the other hand, if you want to replace just, say, font **B**, you have to replace the font at position 2 (with a call to '.fp 2 . . .').

Macros

)E *level text*

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

contents.

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

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

AE Abstract end, see **AS**.

AF [*name-of-firm*]
Author's firm, should be called before **AU**, see also **COVER**.

AL [*type* [*text-indent* [**1**]]]
Start auto-increment list. Items are numbered beginning with one. The*type* argument controls the format of numbers.

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

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

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

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

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

Arg	Placement
0	Abstract is printed on page 1 and on the cover sheet if used in the released-paper style (**MT 4**), otherwise it is printed on page 1 without a cover sheet.
1	Abstract is only printed on the cover sheet (**MT 4** only).
2	Abstract is printed only on the cover sheet (other than **MT 4** only). The cover sheet is printed without a need for **CS**.

An abstract is not printed at all in external letters (**MT 5**). The*indent* par ameter controls the indentation of both margins, otherwise normal text indentation is used.

AST [*title*]
Abstract title. Default is 'ABSTRACT'. Sets the text above the abstract text.

AT *title1* [*title2* [. . .]]
Author's title. **AT** must appear just after each **AU**. The title shows up after the name in the signature block.

AU [*name* [*initials* [*loc* [*dept* [*ext* [*room* [*arg* [*arg* [*arg*]]]]]]]]]
Author information. Specifies the author of the memo or paper, and is printed on the cover sheet and on other similar places. **AU** must not appear before **TL**. The author information can contain initials, location, department, telephone extension, room number or name and up to three extra arguments.

AV [*name* [**1**]]
Approval signature. Generates an approval line with place for signature and date. The string 'APPROVED:' can be changed with variable **Letapp**; it is replaced with an empty lin if there is a second argument. The string 'Date' can be changed with variable **Letdate**.

AVL [*name*]
Letter signature. Generates a line with place for signature.

B [*bold-text* [*prev-font-text* [*bold* [. . .]]]]
Begin boldface. No limit on the number of arguments. All arguments are concatenated to one word; the first, third and so on is printed in boldface.

B1 Begin box (as the ms macro). Draws a box around the text. The text is indented one character, and the right margin is one character shorter.

B2 End box. Finishes the box started with **B1**.

BE End bottom block, see **BS**.

BI [*bold-text* [*italic-text* [*bold-text* [. . .]]]]
Bold-italic. No limit on the number of arguments, see **B**.

BL [*text-indent* [**1**]]
Start bullet list. Initializes a list with a bullet and a space in the beginning of each list item (see **LI**). *text-indent* overrides the default indentation of the list items set by number register **Pi**. A third argument prohibits printing of a blank line before each item.

BR [*bold-text* [*roman-text* [*bold-text* [. . .]]]]
Bold-roman. No limit on the number of arguments.

BS Bottom block start. Begins the definition of a text block which is printed at the bottom of each page. The block ends with **BE**.

BVL *text-indent* [*mark-indent* [**1**]]
Start of broken variable-item list. Broken variable-item list has no fixed mark, it assumes that every **LI** has a mark instead. The text always begins at the next line after the mark. *te xt-indent* sets the indentation to the text, and *mark-indent* the distance from the current indentation to the mark. A third argument prohibits printing of a blank line before each item.

COVER [*arg*]

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

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

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

COVEND

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

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

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

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

De register

0 Nothing special, this is the default.

1 A page eject occurs after each printed display, giving only one display per page and no text following it.

Df register

0 Displays are printed at the end of each section (when section-page numbering is active) or at the end of the document.

1 A new display is printed on the current page if there is enough space, otherwise it is printed at the end of the document.

2 One display is printed at the top of each page or column (in multi-column mode).

3 Print one display if there is enough space for it, otherwise it is printed at the top of the next page or column.

4 Print as many displays as possible in a new page or column. A page break occurs between each display if **De** is not zero.

5 Fill the current page with displays and the rest beginning at a new page or column. (This is the default.) A page break occurs between each display if **De** is not zero.

DL [*text-indent* [**1** [**1**]]]
Dash list start. Begins a list where each item is printed after a dash. *text-indent* changes the default indentation of the list items set by number register **Pi**. A second argument prevents an empty line between each list item. See **LI**. A third argument prohibits printing of a blank line before each item.

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

format

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

The values 'L', 'I', 'C', and 'CB' can also be specified as '0', '1', '2', and '3', respectively, for compatibility reasons.

fill

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

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

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

EC [*title* [*override* [*flag* [*refname*]]]]
Equation title. Sets a title for an equation. The *override* argument changes the numbering.

flag

none	*override* is a prefix to the number.
0	*override* is a prefix to the number.
1	*override* is a suffix to the number.
2	*override* replaces the number.

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

The string **Le** controls the title of the List of Equations; default is 'LIST OF EQUATIONS'. The List of Equations is only printed if number register **Le** is 1. The default is 0. The string **Liec** contains the word 'Equation', which is printed

before the number. If*refname* is used, then the equation n umber is saved with **.SETR**, and can be retrieved with '**.GETST** *refname*'.

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

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

This macro defines string **EOPef**.

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

This macro defines string **TPeh**.

EN Equation end, see **EQ**.

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

Strings available to EOP

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

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

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

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

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

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

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

FD [*arg* [**1**]]
Footnote default format. Controls the hyphenation (hyphen), right margin justification (adjust), and indentation of footnote text (indent). It can also change the label justification (ljust).

arg	hyphen	adjust	indent	ljust
0	no	yes	yes	left
1	yes	yes	yes	left
2	no	no	yes	left
3	yes	no	yes	left
4	no	yes	no	left
5	yes	yes	no	left
6	no	no	no	left
7	yes	no	no	left
8	no	yes	yes	right
9	yes	yes	yes	right
10	no	no	yes	right
11	yes	no	yes	right

An argument greater than or equal to 11 is considered as value 0. Default for **mm** is 10.

FE Footnote end.

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

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

FS [*label*]
Footnote start. The footnote is ended by **FE**. By default, footnotes are automatically numbered; the number is available in string **F**. Just add ***F** in the text. By adding *label*, it is possible to have other number or names on the footnotes. Footnotes in displays are now possible. An empty line separates footnotes; the height of the line is controlled by number register **Fs**, default value is 1.

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

GETPN *refname* [*varname*]

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

GETR *refname*

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

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

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

GETST *refname* [*varname*]

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

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

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

Page ejection before heading

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

Heading break level

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

Heading space level

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

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

Post-heading indent

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

Hi

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

Centered section headings

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

Font control of the heading

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

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

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

I I I I I I I I I I I I I I

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

Point size control

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

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

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

Heading counters

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

Automatic table of contents

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

Special control of the heading, user-defined macros

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

HX *level rlevel heading-text*

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

}0 (string)

Contains the heading mark plus two spaces if *rlevel* is nonzero, otherwise empty.

;0 (register)

Contains the position of the text after the heading. 0 means that the text should follow the heading on the same line,

1 means that a line break should occur before the text, and 2 means that a blank line should separate the heading and the text.

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

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

HY *dlevel rlevel heading-text*
This macro is called after size and font calculations and might be used to change indentation.

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

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

HM [*arg1* [*arg2* [. . . [*arg14*]]]]
Heading mark style. Controls the type of marking for printing of the heading counters. Default is 1 for all levels.

Argument

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

HU *heading-text*
Unnumbered section header. **HU** behaves like **H** at the level in number register **Hu**. See**H**.

HX *dlevel rlevel heading-text*
User-defined heading exit. Called just before printing the header. See**H**.

HY *dlevel rlevel heading-text*
User-defined heading exit. Called just before printing the header. See**H**.

HZ *dlevel rlevel heading-text*
User-defined heading exit. Called just after printing the header. See**H**.

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

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

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

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

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

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

type

N Page numbers
H Header marks
B Both page numbers and header marks, separated with a tab character.

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

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

IND *arg1* [*arg2* [. . .]]
Write a line in the index file selected by **INITI** with all arguments and the page number or header mark separated by tabs.

Examples

arg1\tpage number
arg1\targ2\tpage number
arg1\theader mark
arg1\tpage number\theader mark

INDP
Print the index by running the command specified by string variable **Indcmd**, which has 'sort –t\t' as the default value. **INDP** reads the output from the command to form the index, by default in two columns (this can be changed by

defining **TYIND**). The index is printed with string variable **Index** as header, default is 'INDEX'. One-column processing is reactivated after the list. **INDP** calls the user-defined macros **TXIND**, **TYIND**, and **TZIND** if defined. **TXIND** is called before printing the string 'INDEX', **TYIND** is called instead of printing 'INDEX', and **TZIND** is called after the printing and should take care of restoring to normal operation again.

ISODATE [**0**]

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

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

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

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

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

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

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

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

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

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

type

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

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

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

LC [*list-level*]

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

LE [**1**]

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

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

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

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

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

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

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

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

LT [*arg*]

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

Arg	**Style**
BL	Blocked. Date line, return address, writer's address and closing begins at the center of the line. All other lines begin at the left margin.
SB	Semi-blocked. Same as blocked, except that the first line in every paragraph is indented five spaces.
FB	Full-blocked. All lines begin at the left margin.
SP	Simplified. Almost the same as the full-blocked style. Subject and the writer's identification are printed in all-capital.

LO *type* [*arg*]

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

CN	Confidential notation. Prints 'CONFIDENTIAL' on the second line below the date line. Any argument replaces 'CONFIDENTIAL'. See also string variable **LetCN**.
RN	Reference notation. Prints 'In reference to:' and the argument two lines below the date line. See also string variable **LetRN**.

AT Attention. Prints ‘ATTENTION:’ and the argument below the inside address. See also string variable **LetAT**.

SA Salutation. Prints ’To Whom It May Concern:’ or the argument if it was present. The salutation is printed two lines below the inside address. See also string variable **LetSA**.

SJ Subject line. Prints the argument as subject prefixed with ‘SUBJECT:’ two lines below the inside address, except in letter type ‘SP’, where the subject is printed in all-capital without any prefix. See also string variable **LetSJ**.

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

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

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

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

MT [*arg* [*addressee*]]

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

arg

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

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

MOVE *y-pos* [*x-pos* [*line-length*]]

Move to a position, setting page offset to *x-pos*. If*line-length* is not giv en, the difference between current and new page offset is used. Use **PGFORM** without arguments to return to normal.

MULB *cw1 space1* [*cw2 space2* [*cw3* . . .]]

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

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

MULE
End the multi-column mode and print the columns.

nP [*type*]
Print numbered paragraph with header level two. See**.P**.

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

NS [*arg* [**1**]]
Print different types of notations. The argument selects between the predefined type of notations. If the second argument is available, then the argument becomes the entire notation. If the argument doesn't select a predefined type, it is printed as 'Copy (*arg*) to'. It is possible to add more standard notations, see the string variables **Letns** and **Letnsdef**.

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

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

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

This macro defines string **EOPof**.

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

This macro defines string **TPoh**.

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

P [*type*]

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

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

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

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

Set line length, page length, and/or page offset. This macro can be used for special formatting, like letter heads and other. It is normally the first command in a file, though it is not necessary. **PGFORM** can be used without arguments to reset everything after a **MOVE** call. A line break is done unless the fourth argument is given. This can be used to avoid the page number on the first page while setting new width and length. (It seems as if this macro sometimes doesn't work too well. Use the command-line arguments to change line length, page length, and page offset instead.)

PGNH

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

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

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

PE Picture end. Ends a picture for **pic**(1).

PF [*arg*]

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

This macro defines string **EOPf**.

PH [*arg*]

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

"'*left-part*'*center-part*'*right-part*'"

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

"”- % -”"

which gives the page number between two dashes.

This macro defines string **TPh**.

PS Picture start (from pic). Begins a picture for **pic**(1).

PX Page header user-defined exit. This macro is called just after the printing of the page header in *no-space* mode.

R Roman. Return to roman font, see also **I**.

RB [*roman-text* [*bold-text* [*roman-text* [. . .]]]]
Roman-bold. Even arguments are printed in roman, odd in boldface. See**I**.

RD [*prompt* [*diversion* [*string*]]]
Read from standard input to diversion and/or string. The text is saved in a diversion named *diversion*. Recall the text by writing the name of the diversion after a dot on an empty line. A string is also defined if*string* is giv en. *Diversion* and/or *prompt* can be empty ("").

RF Reference end. Ends a reference definition and returns to normal processing. See **RS**.

RI [*roman-text* [*italic-text* [*roman-text* [. . .]]]]
Print even arguments in roman, odd in italic. See **I**.

RL [*text-indent*[**1**]]
Reference list start. Begins a list where each item is preceded with an automatically incremented number between square brackets. *text-indent* changes the default indentation.

RP [*arg1* [*arg2*]]
Produce reference page. This macro can be used if a reference page is wanted somewhere in the document. It is not needed if **TC** is used to produce a table of contents. The reference page is then printed automatically.

The reference counter is not reset if *arg1* is 1.

arg2 tells **RP** whether to eject a page or not.

arg2

0 The reference page is printed on a separate page.
1 Do not eject page after the list.
2 Do not eject page before the list.
3 Do not eject page before and after the list.

The reference items are separated by a blank line. Setting number register **Ls** to 0 suppresses the line.

The string **Rp** contains the reference page title and is set to ‘REFERENCES’ by default. The number register **Rpe** holds the default value for the second argument of **RP**; it is initially set to 0.

RS [*string-name*]
Begin an automatically numbered reference definition. Put the string ***(Rf** where the reference mark should be and write the reference between **RS**/**RF** at next new line after the reference mark. The reference number is stored in

number register **:R**. If*string-name* is giv en, a string with that name is defined and contains the current reference mark. The string can be referenced as ***[***string-name***]** later in the text.

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

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

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

SG [*arg* [**1**]]
Signature line. Prints the authors name(s) after the formal closing. The argument is appended to the reference data, printed at either the first or last author. The reference data is the location, department, and initials specified with **.AU**. It is printed at the first author if the second argument is given, otherwise at the last. No reference data is printed if the author(s) is specified through **.WA**/**.WE**. See section **INTERNALS**.

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

SM *string1* [*string2* [*string3*]]
Make a string smaller. If*string2* is giv en, *string1* is made smaller and *string2* stays at normal size, concatenated with *string1*. With three arguments, everything is concatenated, but only *string2* is made smaller.

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

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

TB [*title* [*override* [*flag* [*refname*]]]]
Table title. The arguments are the same as for **EC**. **TB** uses the number register **Tb** as a counter. The string**Lt** controls the title of the List of T ables; default value is 'LIST OF TABLES'. The List of Tables is only printed if number register **Lt** is 1, which is the default. The string**Litb** contains the w ord 'TABLE', which is printed before the number.

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

TC [*slevel* [*spacing* [*tlevel* [*tab* [*h1* [*h2* [*h3* [*h4* [*h5*]]]]]]]]]
Table of contents. This macro is normally used as the last line of the document. It generates a table of contents with headings up to the level controlled

by number register **Cl**. Note that**Cl** controls the sa ving of headings, it has nothing to do with **TC**. Headings with a level less than or equal to *slevel* get *spacing* number of lines before them. Headings with a level less than or equal to *tlevel* have their page numbers right–justified with dots or spaces separating the text and the page number. Spaces are used if*tab* is g reater than zero, dots otherwise. Other headings have the page number directly at the end of the heading text (*ragged-right*).

The rest of the arguments is printed, centered, before the table of contents.

The user-defined macros **TX** and **TY** are used if **TC** is called with at most four arguments. **TX** is called before the printing of the string 'CONTENTS', and **TY** is called instead of printing 'CONTENTS'.

Equivalent macros can be defined for list of figures, tables, equations and exhibits by defining **TX***xx* or **TY***xx*, where *xx* is 'Fg', 'TB', 'EC', or 'EX', respectively.

String **Ci** can be set to control the indentations for each heading-level. It must be scaled, like

```
.ds Ci .25i .5i .75i 1i 1i
```

By default, the indentation is controlled by the maximum length of headings in each level.

The string variables **Lifg**, **Litb**, **Liex**, **Liec**, and **Licon** contain 'Figure', 'TABLE', 'Exhibit', 'Equation', and 'CONTENTS', respectively. These can be redefined to other languages.

TE Table end. See **TS**.

TH [**N**]
Table header. See**TS**. **TH** ends the header of the tab le. This header is printed again if a page break occurs. Argument 'N' isn't implemented yet.

TL [*charging-case-number* [*filing-case-number*]]
Begin title of memorandum. All text up to the next **AU** is included in the title. *charging-case-number* and *filing-case-number* are saved for use in the front page processing.

TM [*num1* [*num2* [. . .]]]
Technical memorandum numbers used in **.MT**. An unlimited number of arguments may be given.

TP Top-of-page user-defined macro. This macro is called instead of the normal page header. It is possible to get complete control over the header. Note that the header and the footer are printed in a separate environment. Line length is preserved, though. See **EOP**.

strings available to TP

TPh	argument of **PH**
TPeh	argument of **EH**
TPoh	argument of **OH**

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

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

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

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

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

VERBOFF
End verbatim output.

VL *text-indent* [*mark-indent* [**1**]]
Variable-item list. It has no fixed mark, it assumes that every **LI** has a mark instead. *text-indent* sets the indent to the text, and *mark-indent* the distance from the current indentation to the mark. A third argument prohibits printing of a blank line before each item.

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

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

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

WC [*format1*] [*format2*] [. . .]
Footnote and display width control.

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

Strings used in mm

App A string containing the word 'APPENDIX'.

Apptxt
The current appendix text.

EM Em dash string

H1txt
Updated by **.H** and **.HU** to the current heading text. Also updated in table of contents & friends.

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

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

Index
Contains the string 'INDEX'.

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

Lifg String containing 'Figure'.

Litb String containing 'TABLE'.

Liex String containing 'Exhibit'.

Liec
String containing 'Equation'.

Licon
String containing 'CONTENTS'.

Lf Contains the string 'LIST OF FIGURES'.

Lt Contains the string 'LIST OF TABLES'.

Lx Contains the string 'LIST OF EXHIBITS'.

Le Contains the string 'LIST OF EQUATIONS'.

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

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

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

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

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

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

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

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

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

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

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

"MO1 – MO12"
Strings containing the month names 'January' through 'December'.

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

Rp Contains the string ‘REFERENCES’.

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

Value	Meaning
co	Table of contents
fg	List of figures
tb	List of tables
ec	List of equations
ex	List of exhibits
ap	Appendix

Tm Contains the string ‘\(tm’, the trade mark symbol.

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

Number variables used in mm

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

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

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

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

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

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

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

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

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

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

Fs Footnote spacing. Default is 1.

"H1 – H7"
Heading counters

H1dot
Append a dot after the level-one heading number if value is greater than zero. Default is 1.

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

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

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

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

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

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

Hps1
Number of lines preceding **.H** if the heading level is greater than **Hps**. Value is in units, default is 0.5.

Hps2
Number of lines preceding **.H** if the heading level is less than or equal to **Hps**. Value is in units, default is 1.

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

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

Ht Heading numbering type.

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

Default is 0.

Hu Unnumbered heading level. Default is 2.

Hy Hyphenation status of text body.

0 no hyphenation
1 hyphenation on, set to value 6

Default is 0.

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

L Page length, only for command-line settings.

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

Lf, **Lt**, **Lx**, **Le**
Enable (1) or disable (0) the printing of List of figures, List of tables, List of exhibits and List of equations, respectively. Default values are Lf=1, Lt=1, Lx=1, and Le=0.

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

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

0 no space
1 emit space

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

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

N Page numbering style.

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

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

Np A flag to control whether paragraphs are numbered.

0 not numbered
1 numbered in first-level headings.

Default is 0.

O Page offset, only for command-line settings.

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

0 ". "
1 " - "

Default is 0.

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

Pi Paragraph indentation. Default is 5.

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

0 Point size only changes to the current setting when **.PH**, **.PF**, **.OH**, **.EH**, **.OF**, or **.OE** is executed.
1 Point size changes after every **.S**. This is the default.

Ps Paragraph spacing. Default is 1.

Pt Paragraph type.

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

Default is 0.

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

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

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

Si Display indentation. Default is 5.

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

W Line length, only for command-line settings.

.mgm
Always 1.

INTERNALS

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

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

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

let@sg_*type name title n flag* [*arg1* [*arg2* [. . .]]]
.SG is calling this macro only for letters; memorandums have its own processing. *name* and *title* are specified through **.WA**/**.WB**. *n* is the counter, 1-max, and *flag* is true for the last name. Any other argument to **.SG** is appended.

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

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

FILES

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

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

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

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

AUTHORS

The GNU version of the *mm* macro package was written by Jörgen Hägg of Lund, Sweden.

SEE ALSO

groff(1), **troff**(1), **tbl**(1), **pic**(1), **eqn**(1)
groff_mmse(7)

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*g roff*(1) man page, section “Installation Directories”, for their location.

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

NAME

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

SYNOPSIS

pdfmom [**–Tps**[pdfroff options]] [groff options] *files . . .*
groff [**–mom**] *files . . .*
groff [**–m mom**] *files . . .*

CALLING MOM

mom is a macro set for **groff**, designed primarily to format documents for *PDF* and *PostScript* output. **mom** provides two categories of macros: macros for typesetting, and macros for document processing. The typesetting macros provide access to groff’s typesetting capabilities in ways that are simpler to master than groff’s primitives. The document processing macros provide highly customizable markup tags that allow the user to design and output professional-looking documents with a minimum of typesetting intervention.

Files processed with **pdfmom**(1) with or without the –T*ps* option, produce *PDF* documents. The documents include a*PDF* outline that appears in the ‘Contents’ panel of document viewers, and may contain clickable internal and external links. When –T*ps* is absent, **groff’s** native *PDF* driver, **gropdf**, is used to generate the output. When given, the output is still *PDF*, but processing is passed over to **pdfroff**, which uses **groff’s** PostScript driver, **grops**. Not all*PDF* f eatures are available when –T*ps* is given; its primary use is to allow processing of files with embedded *PostScript* images.

Files processed with **groff –mom** (or **–m** *mom) produce PostScript* output by default. **mom** comes with her own very complete documentation in *HTML* format. A separate *PDF manual*, *Producing PDFs* with groff and **mom**, covers full **mom** or *PDF* usage.

FILES

om.tmac
– the main macro file

mom.tmac
– a wrapper file that calls om.tmac directly.

/usr/share/doc/groff-1.22.3.rc1.24-ea225/html/mom/toc.html
– entry point to the HTML documentation

/usr/share/doc/groff-1.22.3.rc1.24-ea225/pdf/mom–pdf.pdf
– the PDF manual, *Producing PDFs with groff and mom*

/usr/share/doc/groff-1.22.3.rc1.24-ea225/examples/mom/*.mom
– example files using mom

DOCUMENTATION IN ALPHABETICAL ORDER

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

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

/usr/share/doc/groff-1.22.3.rc1.24-ea225/html/mom/toc.html
– entry point to the HTML documentation That document is quite good for beginners, but other users should be happy to have some documentation in reference style.

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

QUICK REFERENCE

Quick Reference of Inline Escape Sequences in alphabetical Order

***[**<*colorname*>**]**
begin using an initialized colour inline

[BCK** *n]**
move backwards in a line

***[BOLDER]**
invoke pseudo bold inline (related to macro **.SETBOLDER**)

***[BOLDERX]**
off pseudo bold inline (related to macro **.SETBOLDER**)

[BU** *n]**
move characters pairs closer together inline (related to macro **.KERN**)

***[COND]**
invoke pseudo condensing inline (related to macro **.CONDENSE**)

***[CONDX]**
off pseudo condensing inline (related to macro **.CONDENSE**)

[CONDSUP]**. . .[CONDSUPX]**
pseudo-condensed superscript

***[DOWN** *n*]
temporarily move downwards in a line

***[EN-MARK]**
mark initial line of a range of line numbers (for use with line numbered endnotes)

***[EXT]**
invoke pseudo extending inline (related to macro **.EXTEND**)

***[EXTX]**
off pseudo condensing inline (related to macro **.EXTEND**)

[EXTSUP]**...[EXTSUPX]**
pseudo extended superscript

***[FU** *n*]
move characters pairs further apart inline (related to macro **.KERN**)

***[FWD** *n*]
move forward in a line

***[LEADER]**
insert leaders at the end of a line

***[RULE]**
draw a full measure rule

***[SIZE** *n*]
change the point size inline (related to macro **.PT_SIZE**)

***[SLANT]**
invoke pseudo italic inline (related to macro **.SETSLANT**)

***[SLANTX]**
off pseudo italic inline (related to macro **.SETSLANT**)

[ST**<*n*>**]**...[ST**<*n*>**X]**
string tabs (mark tab positions inline)

[SUP]**...[SUPX]**
superscript

***[TB+]**
inline escape for **.TN** (*Tab Next*)

[UL]**...[ULX]**
invoke underlining inline (fixed width fonts only)

***[UP** *n*]
temporarily move upwards in a line

Quick Reference of Macros in alphabetical Order

.AUTOLEAD
set the linespacing relative to the point size

.B_MARGIN
set a bottom margin

.BR break a justified line

.CENTER
set line-by-line quad centre

.CONDENSE
set the amount to pseudo condense

.EL break a line without advancing on the page

.EXTEND
set the amount to pseudo extend

.FALLBACK_FONT
establish a fallback font (for missing fonts)

.FAM
alias to **.FAMILY**

.FAMILY *<family>*
set the *family type*

.FT set the font style (roman, italic, etc.)

.HI [*<measure>* **]**
hanging indent

.HY automatic hyphenation on/off

.HY_SET
set automatic hyphenation parameters

.IB [*<left measure> <right measure>* **]**
indent both

.IBX [CLEAR]
exit indent both

.IL [*<measure>* **]**
indent left

.ILX [CLEAR]
exit indent left

.IQ [CLEAR]
quit any/all indents

.IR [*<measure>* **]**
indent right

.IRX [CLEAR]
exit indent right

.JUSTIFY
justify text to both margins

.KERN
automatic character pair kerning on/off

.L_MARGIN
set a left margin (page offset)

.LEFT
set line-by-line quad left

.LL set a line length

.LS set a linespacing (leading)

.PAGE
set explicit page dimensions and margins

.PAGEWIDTH
set a custom page width

.PAGELENGTH
set a custom page length

.PAPER *<paper_type>*
set common paper sizes (letter, A4, etc)

.PT_SIZE
set the point size

.QUAD
"justify" text left, centre, or right

.R_MARGIN
set a right margin

.RIGHT
set line-by-line quad right

.SETBOLDER
set the amount of emboldening

.SETSLANT
set the degree of slant

.SPREAD
force justify a line

.SS set the sentence space size

.T_MARGIN
set a top margin

.TI [*<measure>* **]**
temporary left indent

.WS set the minimum word space size

DOCUMENTATION OF DETAILS

Details of Inline Escape Sequences in alphabetical Order

[<colorname>***]**
begin using an initialized colour inline

[BCK** *n]**
move wards in a line

***[BOLDER]**

***[BOLDERX]**

Emboldening on/off

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

Alternatively, if you wanted the whole line emboldened, you should do
Once ***[BOLDER]** is invoked, it remains in effect until turned off. Note: If you're using the document processing macros with **.PRINTSTYLE TYPEWRITE**, **mom** ignores ***[BOLDER]** requests.

[BU** *n]**

move characters pairs closer together inline (related to macro **.KERN**)

***[COND]**

***[CONDX]**

Pseudo-condensing on/off

***[COND]** begins pseudo-condensing type. ***[CONDX]** turns the feature off. Both are inline escapes, therefore they should not appear as separate lines, but rather be embedded in text lines, like this:
***[COND]** remains in effect until you turn it off with ***[CONDX]**. IMPORTANT: You must turn ***[COND]** off before making any changes to the point size of your type, either via the **.PT_SIZE** macro or with the **s** inline escape. If you wish the new point size to be pseudo-condensed, simply reinvoke ***[COND]** afterwards. Equally, ***[COND]** must be turned off before changing the condense percentage with **.CONDENSE**.

Note: If you're using the document processing macros with **.PRINTSTYLE TYPEWRITE**, **mom** ignores ***[COND]** requests.

[CONDSUP]**. . .[CONDSUPX]**

pseudo-condensed superscript

[DOWN** *n]**

temporarily move downwards in a line

***[EN-MARK]**

mark initial line of a range of line numbers (for use with line numbered endnotes)

***[EXT]**

***[EXTX]**

Pseudo-extending on/off

***[EXT]** begins pseudo-extending type. ***[EXTX]** turns the feature off. Both are inline escapes, therefore they should not appear as separate lines, but rather be embedded in text lines, like this:
***[EXT]** remains in effect until you turn it off with ***[EXTX]**.

IMPORTANT: You must turn ***[EXT]** off before making any changes to the point size of your type, either via the **.PT_SIZE** macro or with the **s** inline escape. If you wish the new point size to be *pseudo-extended*, simply reinvoke ***[EXT]** afterwards. Equally, ***[EXT]** must be turned off before changing the extend percentage with **.EXTEND**. Note: If you are using the document processing macros with **.PRINTSTYLE TYPEWRITE**, **mom** ignores ***[EXT]** requests.

[EXTSUP]**. . .[EXTSUPX]**
pseudo extended superscript

[FU** *n]**
move characters pairs further apart inline (related to macro **.KERN**)

[FWD** *n]**
move forward in a line

***[LEADER]**
insert leaders at the end of a line

***[RULE]**
draw a full measure rule

[SIZE** *n]**
change the point size inline (related to macro **.PT_SIZE**)

***[SLANT]**

***[SLANTX]**
Pseudo italic on/off

***[SLANT]** begins *pseudo-italicizing type*. ***[SLANTX]** turns the feature off. Both are *inline escapes*, therefore they should not appear as separate lines, but rather be embedded in text lines, like this:
Alternatively, if you wanted the whole line *pseudo-italicized*, you'd do

Once ***[SLANT]** is invoked, it remains in effect until turned off. Note: If you're using the document processing macros with **.PRINTSTYLE TYPEWRITE**, **mom** underlines pseudo-italics by default. To change this behaviour, use the special macro **.SLANT_MEANS_SLANT**.

[ST**<*number*>**]**. . .[ST**<*number*>**X]**
Mark positions of string tabs

The *quad* direction must be **LEFT** or **JUSTIFY** (see **.QUAD** and **.JUSTIFY**) or the *no-fill mode* set to **LEFT** in order for these inlines to function properly. Please see *IMPORTANT*, below. String tabs need to be marked off with inline escapes before being set up with the **.ST** macro. Any input line may contain string tab markers. <*number*>, above, means the numeric identifier of the tab.

The following shows a sample input line with string tab markers.
String *tab 1* begins at the start of the line and ends after the word *time*. String *tab 2* starts at *good* and ends after *men*. *Inline escapes* (e.g. *font* or *point size changes*, or horizontal movements, including padding) are taken into account when **mom** determines the *position* and *length* of *string tabs*.

Up to nineteen string tabs may be marked (not necessarily all on the same line, of course), and they must be numbered between 1 and 19. Once string tabs have been marked in input lines, they have to be *set* with **.ST**, after which they may be called, by number, with **.TAB**.

Note: Lines with string tabs marked off in them are normal input lines, i.e. they get printed, just like any input line. If you want to set up string tabs without the line printing, use the **.SILENT** macro. *IMPORTANT:* Owing to the way **groff** processes input lines and turns them into output lines, it is not possible for **mom** to *guess* the correct starting position of string tabs marked off in lines

that are centered or set flush right.

Equally, she cannot guess the starting position if a line is fully justified and broken with **.SPREAD**. In other words, in order to use string tabs, **LEFT** must be active, or, if **.QUAD LEFT** or **JUSTIFY** are active, the line on which the *string tabs* are marked must be broken *manually* with **.BR** (but not **.SPREAD**).

To circumvent this behaviour, I recommend using the **PAD** to set up string tabs in centered or flush right lines. Say, for example, you want to use a *string tab* to *underscore* the text of a centered line with a rule. Rather than this,

```
.CENTER
\*[ST1]A line of text\*[ST1X]\c
.EL
.ST 1
.TAB 1
.PT_SIZE 24
.ALD 3p
\*[RULE]
.RLD 3p
.TQ
```

you should do:

```
.QUAD CENTER
.PAD "#\*[ST1]A line of text\*[ST1X]#"
.EL
.ST 1
.TAB 1
.PT_SIZE 24
.ALD 3p
\*[RULE] \" Note that you can't use \*[UP] or \*[DOWN] with \*[RULE]
.RLD 3p
.TQ
```

[SUP]**...[SUPX]**
superscript

***[TB+]**
Inline escape for **.TN** (*Tab Next*)

[UL]**...[ULX]**
invoke underlining inline (fixed width fonts only)

[UP** *n]**
temporarily move upwards in a line

Details of Macros in alphabetical Order

.AUTOLEAD
set the linespacing relative to the point size

.B_MARGIN *<bottom margin>*
Bottom Margin
Requires a unit of measure

.B_MARGIN sets a nominal position at the bottom of the page beyond which you don't want your type to go. When the bottom margin is reached,**mom** starts a new page. **.B_MARGIN requires a unit of measure.** Decimal fractions are allowed. To set a nominal bottom margin of 3/4 inch, enter

```
.B_MARGIN .75i
```

Obviously, if you haven't spaced the type on your pages so that the last lines fall perfectly at the bottom margin, the margin will vary from page to page. Usually, but not always, the last line of type that fits on a page before the bottom margin causes mom to start a new page.

Occasionally, owing to a peculiarity in *groff*, an extra line will fall below the nominal bottom margin. If you're using the document processing macros, this is unlikely to happen; the document processing macros are very hard-nosed about aligning bottom margins. Note: The meaning of**.B_MARGIN** is slightly different when you're using the document processing macros.

.FALLBACK_FONT <*fallback font*> **[ABORT** | **WARN]**

Fallback Font

In the event that you pass an invalid argument to **.FAMILY** (i.e. a non-existent *family*), **mom**, by default, uses the *fallback font*, **Courier Medium Roman** (**CR**), in order to continue processing your file. If you'd prefer another *fallback font*, pass **.FALLBACK_FONT** the full *family+font name* of the *font* you'd like. For example, if you'd rather the *fallback font* were **Times Roman Medium Roman**,

```
.FALLBACK_FONT TR
```

would do the trick.

Mom issues a warning whenever a *font style set* with **.FT** does not exist, either because you haven't registered the style or because the *font style* does not exist in the current *family set* with **.FAMILY**. By default, **mom** then aborts, which allows you to correct the problem. If you'd prefer that **mom** not abort on non-existent *fonts*, but rather continue processing using a *fallback font*, you can pass **.FALLBACK_FONT** the argument **WARN**, either by itself, or in conjunction with your chosen *fallback font***.**

Some examples of invoking **.FALLBACK_FONT**:

.FALLBACK_FONT WARN

mom will issue a warning whenever you try to access a non-existent *font* but will continue processing your file with the default *fallback font*, **Courier Medium Roman**.

.FALLBACK_FONT TR WARN

mom will issue a warning whenever you try to access a non-existent *font* but will continue processing your file with a *fallback font* of **Times Roman Medium Roman**; additionally, **TR** will be the *fallback font* whenever you try to access a *family* that does not exist.

.FALLBACK_FONT TR ABORT

mom will abort whenever you try to access a non-existent **font**, and will use the *fallback font* **TR** whenever you try to access a *family* that does not exist. If, for some reason, you want to revert to **ABORT**, just enter **".FALLBACK_FONT ABORT"** and **mom** will once again abort on *font*

errors.

.FAM *<family>*

Type Family, alias of **.FAMILY**

.FAMILY *<family>*

Type Family, alias **.FAM**

.FAMILY takes one argument: the name of the *family* you want. *Groff* comes with a small set of basic families, each identified by a 1-, 2- or 3-letter mnemonic. The standard families are:

```
A   = Avant Garde
BM  = Bookman
H   = Helvetica
HN  = Helvetica Narrow
N   = New Century Schoolbook
P   = Palatino
T   = Times Roman
ZCM = Zapf Chancery
```

The argument you pass to **.FAMILY** is the identifier at left, above. For example, if you want **Helvetica**, enter

```
.FAMILY H
```

Note: The font macro (**.FT**) lets you specify both the type *family* and the desired font with a single macro. While this saves a few keystrokes, I recommend using **.FAMILY for** *family*, and **.FT for** *font*, except where doing so is genuinely inconvenient. **ZCM**, for example, only exists in one style: **Italic** (**I**).

Therefore,

```
.FT ZCMI
```

makes more sense than setting the *family* to **ZCM**, then setting the *font* to *I*. Additional note: If you are running a version of groff lower than 1.19.2, you must follow all **.FAMILY** requests with a **.FT** request, otherwise **mom** will set all type up to the next **.FT** request in the fallback font.

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

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

However, if the font style does not exist in the new family, **mom** will set all subsequent type in the fallback font (by default, **Courier Medium Roman**) until she encounters a **.FT** request that's valid for the *family*.

For example, assuming you don't have the font **Medium Condensed Roman** (**mom** extension *CD*) in the *Helvetica family*:

```
.FAMILY UN \Universfamily"
.FT CD \MediumCondensed"
<some text> \" Univers Medium Condensed
.FAMILY H \Helveticafamily"
```

```
<more text> \" Courier Medium Roman!
```

In the above example, you must follow **.FAMILY H** with a **.FT** request that's valid for **Helvetica**.

Please see the Appendices, *Adding fonts to groff*, for information on adding fonts and families to groff, as well as to see a list of the extensions **mom** provides to *groff*'s basic **R**, **I**, **B**, **BI** styles. Suggestion: When adding*f amilies to groff*, I recommend following the established standard for the naming families and fonts. For example, if you add the **Garamond** family, name the font files

```
GARAMONDR
GARAMONDI
GARAMONDB
GARAMONDBI
```

GARAMOND then becomes a valid *family name* you can pass to **.FAMILY**. (You could, of course, shorten **GARAMOND** to just **G**, or **GD**.) **R**, **I**, **B**, and **BI** after **GARAMOND** are the *roman*, *italic*, *bold* and *bold-italic* fonts respectively.

.FONT R | **B** | **BI** | *<any other valid font style>*

Alias to **.FT**

.FT R | **B** | **BI** | *<any other valid font style>*

Set font

By default, *groff* permits **.FT** to take one of four possible arguments specifying the desired font:

```
R = (Medium) Roman
I = (Medium) Italic
B = Bold (Roman)
BI = Bold Italic
```

For example, if your *family* is **Helvetica**, entering

```
.FT B
```

will give you the *Helvetica bold font*. If your *family* were **Palatino**, you'd get the *Palatino bold font*.

Mom considerably extends the range of arguments you can pass to **.FT**, making it more convenient to add and access fonts of differing weights and shapes within the same family. Have a look here for a list of the weight/style arguments **mom** allows. Be aware, though, that you must have the fonts, correctly installed and named, in order to use the arguments. (See*Adding f onts to groff* for instructions and information.) Please also read the*ADDITIONAL NO TE* found in the description of the **.FAMILY** macro.

How **mom** reacts to an invalid argument to **.FT** depends on which version of groff you're using. If your *groff version* is greater than or equal to 1.19.2, **mom** will issue a warning and, depending on how you've set up the fallback font, either continue processing using the fallback font, or abort (allowing you to correct the problem). If your *groff version* is less than 1.19.2, **mom** will silently continue processing, using either the fallback font or the font that was in effect prior to the invalid **.FT** call. **.FT** will also accept, as an argument, a full *family* and *font name*.

For example,

```
.FT HB
```

will set subsequent type in *Helvetica Bold*. Ho wever, I strongly recommend

keeping *family* and *font* separate except where doing so is genuinely inconvenient.

For inline control of *fonts*, see *Inline Escapes*, font control.

.HI [*<measure>* **]**

Hanging indent — the optional argument requires a unit of measure.

A hanging indent looks like this:

```
The thousand injuries of Fortunato I had borne as best I
  could, but when he ventured upon insult, I vowed
  revenge.  You who so well know the nature of my soul
  will not suppose, however, that I gave utterance to a
  threat, at length I would be avenged...
```

The first line of text *hangs* outside the *left margin*.

In order to use *hanging indents*, you must first have a *left indent* active (set with either **.IL** or **.IB**). **Mom** will not hang text outside the *left margin set* with **.L_MARGIN** or outside the *left margin* of a *tab*. The first time you invoke **.HI**, you must give it a **measure**. If you want the first line of a paragraph to *hang by*, say, *1 pica*, do

```
.IL 1P
.HI 1P
```

Subsequent invocations of **.HI** do not require you to supply a *measure*; **mom** keeps track of the last measure you gave it.

Generally speaking, you should invoke **.HI** immediately prior to the line you want hung (i.e. without any intervening control lines). And because *hanging indents* affect only one line, there's no need to turn them off. *IMPORTANT:* Unlike **IL**, **IR** and **IB**, measures given to **.HI ,** the measure is treated literally. **.I Recipe:** A numbered list using *hanging indents*

Note: **mom** has macros for setting lists. This recipe exists to demonstrate the use of *hanging indents* only.

```
.PAGE 8.5i 11i 1i 1i 1i 1i
.FAMILY  T
.FT      R
.PT_SIZE 12
.LS      14
.JUSTIFY
.KERN
.SS 0
.IL \w'\0\0.'
.HI \w'\0\0.'
1.\0The most important point to be considered is whether the
answer to the meaning of Life, the Universe, and Everything
really is 42.  We have no-one's word on the subject except
Mr. Adams'.
.HI
2.\0If the answer to the meaning of Life, the Universe,
and Everything is indeed 42, what impact does this have on
the politics of representation?  42 is, after all not a
```

prime number. Are we to infer that prime numbers don't deserve equal rights and equal access in the universe?
.HI
3.\0If 42 is deemed non-exclusionary, how do we present it as the answer and, at the same time, forestall debate on its exclusionary implications?

First, we invoke a left indent with a measure equal to the width of 2 figures spaces plus a period (using the \w inline escape). At this point, the left indent is active; text afterwards would normally be indented. However, we invoke a hanging indent of exactly the same width, which hangs the first line (and first line only!) to the left of the indent by the same distance (in this case, that means "out to the left margin"). Because we begin the first line with a number, a period, and a figure space, the actual text (*The most important point...*) starts at exactly the same spot as the indented lines that follow.

Notice that subsequent invocations of **.HI** don't require a *measure* to be given. Paste the example above into a file and preview it with

```
pdfmom filename.mom | ps2pdf — filename.pdf
```

to see hanging indents in action.

.IB [*<left measure> <right measure>* **]**

Indent both — the optional argument requires a unit of measure

.IB allows you to set or invoke a left and a right indent at the same time. At its first invocation, you must supply a measure for both indents; at subsequent invocations when you wish to supply a measure, both must be given again. As with **.IL** and **.IR**, the measures are added to the values previously passed to the macro. Hence, if you wish to change just one of the values, you must give an argument of zero to the other.

A word of advice: If you need to manipulate left and right indents separately, use a combination of **.IL** and **.IR** instead of **.IB**. You'll save yourself a lot of grief. A*min us sign* may be prepended to the arguments to subtract from their current values. The \w inline escape may be used to specify text-dependent measures, in which case no unit of measure is required. For example,

```
.IB \w'margarine' \w'jello'
```

left indents text by the width of the word *margarine* and right indents by the width of *jello*.

Like **.IL** and **.IR**, **.IB** with no argument indents by its last active values. See the brief explanation of how mom handles indents for more details. *Note:* Calling a *tab* (with **.TAB <n>**) automatically cancels any active indents.

Additional note: Invoking **.IB** automatically turns off **.IL** and **.IR**.

.IL [*<measure>* **]**

Indent left — the optional argument requires a unit of measure

.IL indents text from the left margin of the page, or if you're in a *tab*, from the left edge of the *tab* Once *IL* is on, the *left indent* is applied uniformly to every subsequent line of text, even if you change the line length.

The first time you invoke **.IL**, you must give it a measure. Subsequent invocations with a measure add to the previous measure. A minus sign may be prepended to the argument to subtract from the current measure. The**\w** inline

escape may be used to specify a text-dependent measure, in which case no unit of measure is required. For example,

```
.IL \w'margarine'
```

indents text by the width of the word *margarine*. With no argument,**.IL** indents by its last active value. See the brief explanation of how **mom** handles indents for more details.

Note: Calling a *tab* (with **.TAB <n>**) automatically cancels any active indents. *Additional note:* Invoking **.IL** automatically turns off **IB**.

.IQ [*<measure>* **]**

IQ — quit any/all indents

IMPORTANT NOTE: The original macro for quitting all indents was **.IX**. This usage has been deprecated in favour of **IQ**. **.IX** will continue to behave as before, but **mom** will issue a warning to *stderr* indicating that you should update your documents. As a consequence of this change, **.ILX**, **.IRX** and **.IBX** may now also be invoked as **.ILQ**, **.IRQ** and **.IBQ**. Both forms are acceptable.

Without an argument, the macros to quit indents merely restore your original margins and line length. The measures stored in the indent macros themselves are saved so you can call them again without having to supply a measure. If you pass these macros the optional argument **CLEAR**, they not only restore your original left margin and line length, but also clear any values associated with a particular indent style. The next time you need an indent of the same style, you have to supply a measure again.

.IQ CLEAR, as you'd suspect, quits and clears the values for all indent styles at once.

.IR [*<measure>* **]**

Indent right — the optional argument requires a unit of measure

.IR indents text from the *right margin* of the page, or if you're in a *tab*, from the end of the *tab*.

The first time you invoke **.IR**, you must give it a measure. Subsequent invocations with a measure add to the previous indent measure. A*min us sign* may be prepended to the argument to subtract from the current indent measure. The \w inline escape may be used to specify a text-dependent measure, in which case no *unit of measure* is required. For example,

```
.IR \w'jello'
```

indents text by the width of the word *jello*. With no argument,**.IR** indents b y its last active value. See the brief explanation of how **mom** handles indents for more details.

Note: Calling a *tab* (with **.TAB <n>**) automatically cancels any active indents. *Additional note:* Invoking **.IR** automatically turns off **IB**.

.L_MARGIN *<left margin>*

Left Margin

L_MARGIN establishes the distance from the left edge of the printer sheet at which you want your type to start. It may be used any time, and remains in effect until you enter a new value. Left indents and tabs are calculated from the value you pass to **.L_MARGIN**, hence it's always a good idea to invoke it before starting any serious typesetting. A unit of measure is required. Decimal

fractions are allowed. Therefore, to set the left margin at 3 picas (1/2 inch), you'd enter either

```
.L_MARGIN 3P
```

or

```
.L_MARGIN .5i
```

If you use the macros **.PAGE**, **.PAGEWIDTH** or **.PAPER** without invoking **.L_MARGIN** (either before or afterwards), **mom** automatically sets **.L_MARGIN** to *1 inch*. Note:**.L_MARGIN** beha ves in a special way when you're using the document processing macros.

.MCO

Begin multi-column setting.

.MCO (*Multi-Column On*) is the *macro* you use to begin *multi-column setting*. It marks the current baseline as the top of your columns, for use later with **.MCR**. See the introduction to columns for an explanation of *multi-columns* and some sample input. *Note:* Do not confuse **.MCO** with the **.COLUMNS** macro in the document processing macros.

.MCR

Once you've turned *multi-columns* on (with **.MCO**), **.MCR**, at any time, returns you to the *top of your columns*

.MCX [*<distance to advance below longest column>* **]**

Optional argument requires a unit of measure.

.MCX takes you out of any *tab* you were in (by silently invoking **.TQ**) and advances to the bottom of the longest column. Without an argument, **.MCX** advances *1 linespace* below the longest column.

Linespace, in this instance, is the leading in effect at the moment **.MCX** is invoked. If you pass the *<distance>* argument to **.MCX**, it advances *1 linespace* below the longest column (see above) *PLUS* the distance specified by the argument. The argument requires a unit of measure; therefore, to advance an extra 6 points below where **.MCX** would normally place you, you'd enter

```
.MCX 6p
```

Note: If you wish to advance a precise distance below the baseline of the longest column, use **.MCX** with an argument of **(zero; no** *unit of measure* required) in conjunction with the **.ALD** macro, like this:

```
.MCX 0
.ALD 24p
```

The above advances to precisely *24 points* below the baseline of the longest column.

.NEWPAGE

Whenever you want to start a new page, use **.NEWPAGE**, by itself with no argument. **Mom** will finish up processing the current page and move you to the top of a new one (subject to the top margin set with **.T_MARGIN**).

.PAGE *<width>* **[** *<length>* **[** *<lm>* **[** *<rm>* **[** *<tm>* **[** *<bm>* **]]]]]**

All arguments require a unit of measure *IMPORTANT:* If you're using the document processing macros, **.PAGE** must come after **.START**. Otherwise, it should go at the top of a document, prior to any text. And remember, when

you're using the document processing macros, top margin and bottom margin mean something slightly different than when you're using just the typesetting macros (see Top and bottom margins in document processing).

.PAGE lets you establish paper dimensions and page margins with a single macro. The only required argument is page width. The rest are optional, but they must appear in order and you can't skip over any. *<lm>*, *<rm>*, *<tm>* and *<bm>* refer to the left, right, top and bottom margins respectively. Assuming your page dimensions are 11 inches by 17 inches, and that's all you want to set, enter

```
.PAGE 11i 17i
```

If you want to set the left margin as well, say, at 1 inch, **PAGE** would look like this:

```
.PAGE 11i 17i 1i
```

Now suppose you also want to set the top margin, say, at 1–1/2 inches. *<tm>* comes after *<rm>* in the optional arguments, but you can't skip over any arguments, therefore to set the top margin, you must also give a right margin. The **.PAGE** macro would look like this:

```
.PAGE 11i 17i 1i 1i 1.5i
                  |   |
required right---+   +---top margin
        margin
```

Clearly, **.PAGE** is best used when you want a convenient way to tell **mom** just the dimensions of your printer sheet (width and length), or when you want to tell her everything about the page (dimensions and all the margins), for example

```
.PAGE 8.5i 11i 45p 45p 45p 45p
```

This sets up an 8½ by 11 inch page with margins of 45 points (5/8-inch) all around.

Additionally, if you invoke **.PAGE** with a top margin argument, any macros you invoke after **.PAGE** will almost certainly move the baseline of the first line of text down by one linespace. To compensate, do

```
.RLD 1v
```

immediately before entering any text, or, if it's feasible, make **.PAGE** the last macro you invoke prior to entering text. Please read the*Important note* on page dimensions and papersize for information on ensuring groff respects your **.PAGE** dimensions and margins.

.PAGELENGTH *<length of printer sheet>*

tells **mom** how long your printer sheet is. It works just like **.PAGEWIDTH**.

Therefore, to tell **mom** your printer sheet is 11 inches long, you enter

```
.PAGELENGTH 11i
```

Please read the important note on page dimensions and papersize for information on ensuring groff respects your *PAGELENGTH*.

.PAGEWIDTH *<width of printer sheet>*

The argument to **.PAGEWIDTH** is the width of your printer sheet.

.PAGEWIDTH requires a unit of measure. Decimal fractions are allowed. Hence, to tell **mom** that the width of your printer sheet is 8½ inches, you enter

```
.PAGEWIDTH 8.5i
```

Please read the Important note on page dimensions and papersize for

information on ensuring groff respects your *PAGEWIDTH*.

.PAPER *<paper type>*
provides a convenient way to set the page dimensions for some common printer sheet sizes. The argument*<paper type>* can be one of: **LETTER**, **LEGAL**, **STATEMENT**, **TABLOID**, **LEDGER**, **FOLIO**, **QUARTO**, **EXECUTIVE**, **10x14**, **A3**, **A4**, **A5**, **B4**, **B5**.

.PRINTSTYLE

.PT_SIZE *<size of type in points>*
Point size of type, does not require a *unit of measure*.

.PT_SIZE (*Point Size*) takes one argument: the *size of type* in *points*. Unlike most other macros that establish the *size* or *measure* of something, **.PT_SIZE** does not require that you supply a *unit of measure* since it's a near universal convention that *type size* is measured in *points*. Therefore, to change the *type size* to, say, *11 points*, enter

```
.PT_SIZE 11
```

Point sizes may be *fractional* (e.g. *10.25* or *12.5*). You can prepend a *plus* or a *minus sign* to the argument to **.PT_SIZE**, in which case the *point size* will be changed by *+* or *−* the original value. For example, if the *point size* is *12 ,* and you want *14 ,* you can do

```
.PT_SIZE +2
```

then later reset it to *12* with

```
.PT_SIZE −2
```

The *size of type* can also be changed inline.

Note: It is unfortunate that the **pic** preprocessor has already taken the name, PS, and thus *mom*'s macro for setting *point sizes* can't use it. However, if you aren't using **pic**, you might want to alias **.PT_SIZE** as **.PS**, since there'd be no conflict. For example

```
.ALIAS PS PT_SIZE
```

would allow you to set *point sizes* with **.PS**.

.R_MARGIN *<right margin>*
Right Margin
Requires a unit of measure.

IMPORTANT: **.R_MARGIN**, if used, must come after **.PAPER**, **.PAGEWIDTH**, **.L_MARGIN**, and/or **.PAGE** (if a right margin isn't given to PAGE). The reason is that **.R_MARGIN** calculates line length from the overall page dimensions and the left margin. Obviously, it can't make the calculation if it doesn't know the page width and the left margin.

.R_MARGIN establishes the amount of space you want between the end of typeset lines and the right hand edge of the printer sheet. In other words, it sets the line length. **.R_MARGIN** requires a unit of measure . Decimal fractions are allowed. The line length macro (LL) can be used in place of**.R_MARGIN**. In either case, the last one invoked sets the line length. The choice of which to use is up to you. In some instances, you may find it easier to think of a section of type as having a right margin. In others, giving a line length may make more sense.

For example, if you're setting a page of type you know should have 6-pica margins left and right, it makes sense to enter a left and right margin, like this:

```
.L_MARGIN 6P
.R_MARGIN 6P
```

That way, you don't have to worry about calculating the line length. On the other hand, if you know the line length for a patch of type should be 17 picas and 3 points, entering the line length with LL is much easier than calculating the right margin, e.g.

```
.LL 17P+3p
```

If you use the macros **.PAGE**, **.PAGEWIDTH** or **PAPER** without invoking **.R_MARGIN** afterwards, **mom** automatically sets **.R_MARGIN** to *1 inch*. If you set a line length after these macros (with **.LL**), the line length calculated by **.R_MARGIN** is, of course, overridden. Note:**.R_MARGIN** beha ves in a special way when you're using the document processing macros.

.ST *<tab number>* **L | R | C | J [QUAD]**

After *string tabs* have been marked off on an input line (see ***[ST]...*[STX]**), you need to *set* them by giving them a direction and, optionally, the **QUAD** argument. In this respect,**.ST** is lik e **.TAB_SET** except that you don't have to give **.ST** an indent or a line length (that's already taken care of, inline, by ***[ST]...*[STX]**).

If you want string *tab 1* to be **left**, enter

```
.ST 1 L
```

If you want it to be *left* and *filled*, enter

```
.ST 1 L QUAD
```

If you want it to be justified, enter

```
.ST 1 J
```

.TAB *<tab number>*

After *tabs* have been defined (either with **.TAB_SET** or **.ST**), **.TAB** moves to whatever *tab number* you pass it as an argument.

For example,

```
.TAB 3
```

moves you to *tab 3*.

Note: **.TAB** breaks the line preceding it and advances 1 linespace. Hence,

```
.TAB 1
A line of text in tab 1.
.TAB 2
A line of text in tab 2.
```

produces, on output

```
A line of text in tab 1.

                                A line of text in tab 2.
```

If you want the tabs to line up, use **.TN** (*Tab Next*) or, more conveniently, the inline escape *[TB+]:

```
.TAB 1
A line of text in tab 1.\*[TB+]
A line of text in tab 2.
```

which produces

```
A line of text in tab 1.   A line of text in tab 2.
```

If the text in your tabs runs to several lines, and you want the first lines of each tab to align, you must use the multi-column macros. *Additional note:* Any indents in effect prior to calling a tab are automatically turned off by **TAB**. If you were happily zipping down the page with a left indent of *2 picas* turned on, and you call a *tab* whose indent from the left margin is *6 picas*, your new distance from the *left margin* will be *6 picas*, not l 6 picas plus the 2 pica indent.

Tabs are not by nature columnar, which is to say that if the text inside a *tab* runs to several lines, calling another *tab* does not automatically move to the baseline of the first line in the *previous tab*. To demonstrate:

```
TAB 1
Carrots
Potatoes
Broccoli
.TAB 2
$1.99/5 lbs
$0.25/lb
$0.99/bunch
```

produces, on output

```
Carrots
Potatoes
Broccoli
            $1.99/5 lbs
            $0.25/lb
            $0.99/bunch
```

.TB *<tab number>*

Alias to **.TAB**

.TI [*<measure>* **]**

Temporary left indent — the optional argument requires a *unit of measure*

A temporary indent is one that applies only to the first line of text that comes after it. Its chief use is indenting the first line of paragraphs. (**Mom's .PP** macro, for example, uses a *temporary indent*.)

The first time you invoke **.TI**, you must give it a measure. If you want to *indent* the first line of a paragraph by, say, 2 ems, do

```
.TI 2m
```

Subsequent invocations of **.TI** do not require you to supply a measure; **mom** keeps track of the last measure you gave it.

Because *temporary indents* are temporary, there's no need to turn them off. *IMPORTANT:* Unlike **.IL**, **.IR** and **IB**, measures given to **.TI** are NOT additive. In the following example, the second **".TI 2P"** is exactly *2 picas*.

```
.TI 1P
The beginning of a paragraph. . .
.TI 2P
The beginning of another paragraph. . .
```

.TN Tab Next

Inline escape ***[TB+] TN** moves over to the *next tab* in numeric sequence (*tab n+1*) without advancing on the page. See the*NO TE* in the description of the **.TAB** macro for an example of how **TN** works.

In *tabs* that aren't given the **QUAD** argument when they're set up with **.TAB_SET** or **ST**, you must terminate the line preceding **.TN** with the **\c** inline escape. Conversely, if you did give a **QUAD** argument to **.TAB_SET** or **ST**, the **\c must not be used.** If y ou find remembering whether to put in the **\c** bothersome, you may prefer to use the inline escape alternative to **.TN**, ***[TB**, which works consistently regardless of the fill mode.

Note: You must put text in the input line immediately after **.TN**. Stacking of **.TN**'s is not allowed. In other words, you cannot do

```
.TAB 1
Some text\c
.TN
Some more text\c
.TN
.TN
Yet more text
```

The above example, assuming *tabs* numbered from *1* to *4*, should be entered

```
.TAB 1
Some text\c
.TN
Some more text\c
.TN
\&\c
.TN
Yet more text
```

\& is a zero-width, non-printing character that *groff* recognizes as valid input, hence meets the requirement for input text following **.TN**.

.TQ **TQ** takes you out of whatever *tab* you were in, advances *1 linespace*, and restores the *left margin*, *line length*, *quad direction* and *fill mode* that were in effect prior to invoking any *tabs*.

.T_MARGIN *<top margin>*

Top margin

Requires a unit of measure

.T_MARGIN establishes the distance from the top of the printer sheet at which you want your type to start. It requires a unit of measure, and decimal fractions are allowed. To set a top margin of 2½ centimetres, you'd enter

```
.T_MARGIN 2.5c
```

.T_MARGIN calculates the vertical position of the first line of type on a page by treating the top edge of the printer sheet as a baseline. Therefore,

```
.T_MARGIN 1.5i
```

puts the baseline of the first line of type 1½ inches beneath the top of the page. Note: **.T_MARGIN** means something slightly different when you're using the document processing macros. See Top and bottom margins in document processing for an explanation.

IMPORTANT: **.T_MARGIN** does two things: it establishes the top margin for pages that come after it and it moves to that position on the current page. Therefore, **.T_MARGIN** should only be used at the top of a file (prior to entering text) or after NEWPAGE, like this:

```
.NEWPAGE
.T_MARGIN 6P
<text>
```

AUTHORS

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

SEE ALSO

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

/usr/share/doc/groff-1.22.3.rc1.24-ea225/html/mom/toc.html
– entry point to the HTML documentation

http://www.schaffter.ca/mom/momdoc/toc.html – HTML documentation online

http://www.schaffter.ca/mom/ – the mom macros homepage

BUGS

Please send bug reports to the groff-bug mailing list or directly to the authors.

4.6. ms

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

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

4.6.1. Introduction to ms

The ms macros are the oldest surviving macro package for roff systems.[8] While theman package was intended for brief documents to be perused at a terminal, the ms macros are suitable for longer documents intended for printing and possible publication.

The ms macro package included with groff is a complete re-implementation. Some macros specific to AT&T or Berkeley are not included, while several new commands been introduced. See Differences from AT&T ms.

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

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

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

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

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

4.6.2. General structure of an ms document

The ms macro package expects a certain amount of structure, but not as much as packages such as man or mdoc. The simplest documents can begin with a paragraph macro (such as LP or PP), and consist of text separated by paragraph macros or even blank lines. Longer documents have a structure as follows.

Document type

If you invoke the RP (report) macro on the first line of the document, ms prints the cover page information on its own page; otherwise it prints the information (if any) on the first page with your document text immediately following. Some

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

document types found in AT&T `troff` are specific to AT&T or Berkeley, and are not supported in `groff`.

Format and layout
By setting registers (and one string), you can change your document's type (font and point size), margins, spacing, headers and footers, and footnotes. See Document control settings.

Cover page
A cover page consists of a title, the author's name and institution, an abstract, and the date.[9] See Cover page macros.

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

Table of contents
Longer documents usually include a table of contents, which you can produce by placing the `TC` macro at the end of your document. Printing the table of contents at the end is necessary since GNU `troff`, like its AT&T ancestor, is a single-pass text formatter; it thus cannot determine the page number of each section until that section has been set and output. Since `ms` output is designed for hard copy, you can manually relocate the pages containing the table of contents between the cover page and the body text after printing.[10]

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

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

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

A list of document control registers and strings follows. They are presented in the syntax used to interpolate them.

Margin Settings

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

[9] Actually, only the title is required.

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

Effective: next page.

Default value: 1 i.

`\n[LL]`

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

Effective: next paragraph.

Default: 6 i.

`\n[LT]`

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

Effective: next paragraph.

Default: 6 i.

`\n[HM]`

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

Effective: next page.

Default: 1 i.

`\n[FM]`

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

Effective: next page.

Default: 1 i.

Text Settings

`\n[PS]`

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

Effective: next paragraph.

Default: 10 p.

`\n[VS]`

Defines the space between lines (line height plus leading). If the value is larger than or equal to 1000, divide it by 1000 to get a fractional point size.

Effective: next paragraph.

Default: 12 p.

`\n[HY]`

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

Effective: next paragraph.

Default: 6.

`\*[FAM]`

Defines the font family used to typeset the document.

Effective: next paragraph.

Default: as defined in the output device.

Paragraph Settings

`\n[PI]`
: Defines the initial indentation of a (`PP` macro) paragraph.

  Effective: next paragraph.

  Default: 5 n.

`\n[PD]`
: Defines the space between paragraphs.

  Effective: next paragraph.

  Default: 0.3 v.

`\n[QI]`
: Defines the indentation on both sides of a quoted (`QP`, `QS`, and `QE` macros) paragraph.

  Effective: next paragraph.

  Default: 5 n.

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

  Effective: next paragraph.

  Default: 1.

Section Heading Settings

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

  Effective: next section heading.

  Default: 1 p.

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

  Effective: next section heading.

  Default: 0.

`\n[HORPHANS]`
: Defines the minimum number of lines of an immediately succeeding paragraph that should be kept together with any section heading introduced by the `NH` or `SH` macros. If a section heading is placed close to the bottom of a page, and there is insufficient

space to accommodate both the heading and at least HORPHANS lines of the following paragraph, before an automatic page break, then the page break is forced before the heading.

Effective: next paragraph.

Default: 1.

*[SN-STYLE]
Defines the style used to print section numbers within numbered section headings. See Headings.

Effective: next section heading.

Default: alias of SN-DOT

Footnote Settings

\n[FI]
Defines the footnote indentation.

Effective: next footnote.

Default: 2 n.

\n[FF]
The footnote format:

0 Print the footnote number as a superscript; indent the footnote (default).

1 Print the number followed by a period (like 1.) and indent the footnote.

2 Like 1, without an indentation.

3 Like 1, but print the footnote number as a hanging paragraph.

Effective: next footnote.

Default: 0.

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

Effective: next footnote.

Default: \n[PS] - 2.

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

Effective: next footnote.

Default: \n[FPS] + 2.

\n[FPD]
Defines the footnote paragraph spacing.

Effective: next footnote.

Default: \n[PD] / 2.

`\*[FR]`
: Defines the ratio of the footnote line length to the current line length.

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

  Default: 11/12.

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,[11] an`FL` register w as 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.

`FR` should be used in preference to the old `FL` register in contemporary documents. The footnote line length is effectively computed as '`\n[`*column-width*`] * \*[FR]`'. If an absolute footnote line length is required, recall that arithmetic expressions in `roff` languages are evaluated from left to right.

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

Other Settings

`\n[DD]`
: Sets the vertical spacing before and after a display, a `tbl` table, an `eqn` equation, or a `pic` image.

  Effective: next paragraph.

  Default: 0.5 v.

`\n[MINGW]`
: Defines the minimum width between columns in a multi-column document.

  Effective: next page.

  Default: 2 n.

4.6.4. Cover page macros

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

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

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

`.P1`
: (P-one) Prints the header on page 1. The default is to suppress the header.

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

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

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

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

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

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

...
```

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

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

`.AE`
Ends the abstract.

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

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

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

4.6.5. Body text

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

4.6.5.1. Paragraphs

The following paragraph types are available.

`.PP`
: Sets a paragraph with an initial indentation.

`.LP`
: Sets a paragraph without an initial indentation.

`.QP`
: Sets a paragraph that is indented at both left and right margins by the amount of the register `QI`. The next paragraph or heading returns margins to normal. `QP` inserts vertical space of amount set by register `PD` before the paragraph.

`.QS`
`.QE`

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

`.XP`

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

The following markup uses all four paragraph macros.

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

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

4.6.5.2. Headings

Use headings to create a hierarchical structure for your document. The `ms` macros print headings in **bold** using the same font family and, by default, point size as the body text. Numbered and unnumbered section headings are available. Text lines after heading macros are treated as part of the heading, rendered on the same output line in the same style.

`.NH` *level*
`.NH S`*section-le vel-index* ...

Numbered heading.

The *level* argument instructs `ms` to number sections in the form *x.y.z*, to any depth desired, with the numbering of each level increasing automatically and being reset when a more significant level is increased. "1" is the most significant or coarsest division of the document. Only nonzero values are output.

If you specify heading levels with a gap in an ascending sequence, such as by invoking '`.NH 3`' after '`.NH 1`', `groff ms` emits a warning on the standard error stream.

Alternatively, a first argument of `S` can be given, followed by integral arguments to number the levels of the heading explicitly. Further automatic section numbering, if used, resumes using the specified section numbers as their predecessors.

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 section numbering as follows; the vertical space that normally precedes each section heading is omitted.

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

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

After invocation of `NH`, the assigned section number is made available in the strings `SN-DOT` (as it appears in a printed section heading with default formatting, followed by

a terminating period), and SN-NO-DOT (with the terminating period omitted).

You can control the style used to print section numbers within numbered section 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 section numbers appearing in numbered section headings, you may define the alias as follows.

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

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

.SH [*match-level*]

Unnumbered subheading.

The optional*match-le vel* argument is a GNU extension. It is a number indicating the level of the heading corresponding to the*le vel* argument to NH. Its purpose is to match the point size at which the heading is printed to the size of a numbered heading at the same level when the GROWPS and PSINCR heading size adjustment mechanism is in effect.

If the GROWPS register is set to a value greater than the*le vel* argument to NH or SH, the point size of a heading produced by these macros increases by PSINCR units over the size specified by PS, multiplied by the difference between *level* and GROWPS. The value stored in PSINCR is interpreted in groff basic units; the p scaling factor 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 orphaned section 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 or tbl, pic, or eqn region between the heading and the subsequent paragraph suppresses this grouping.

4.6.5.3. Highlighting

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

`.B [`*txt* `[`*post* `[`*pre*`]]]`
: Sets its first argument in **bold type**. If you specify a second argument, `groff ms` prints it in the previous font after the bold text, with no intervening space (this allows you to set punctuation after the highlighted text without highlighting the punctuation). Similarly, it prints the third argument (if any) in the previous font *before* the first argument. For example,

```
.B foo ) (
```

prints '`(foo)`'.

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

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

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

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

`.CW [`*txt* `[`*post* `[`*pre*`]]]`
: Sets its first argument in a `constant-width` (monospaced) roman typeface. It operates similarly to the `B` macro otherwise. This is a Version 10 Research Unix extension.

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

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

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

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

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

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

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

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

4.6.5.4. Lists

The `IP` macro handles duties for all lists.

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

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

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

The following is an example of a bulleted list.

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

Produces:

A bulleted list:

- lawyers
- guns
- money

The following is an example of a numbered list.

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

Produces:

A numbered list:

1. lawyers

2. guns

3. money

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

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

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

Produces:

A glossary-style list:

lawyers
      Two or more attorneys.

guns Firearms, preferably large-caliber.

money
      Gotta pay for those lawyers and guns!

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

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

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

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

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

4.6.5.5. Indented regions

You may need to indent a section of text while still wrapping and filling.

`.RS`
: Begin a region indented by the amount of the `PI` register, affecting the placement of headings, paragraphs, and displays.

`.RE`
: End the most recent indented region.

You can use `RS`/`RE` regions to line up text under hanging and indented paragraphs. For example, you may wish to nest lists; the input

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

produces

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

as output.

See Displays and keeps, for macros to indent regions with filling disabled.

4.6.5.6. Tab stops

Use the `ta` request to define tab stops as needed. See Tabs and Fields.

`.TA`

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

4.6.5.7. Displays and keeps

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

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

`.DS L`
`.LD`
`.DE`

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

.DS I
.ID
.DE

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

.DS B
.BD
.DE

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

.DS C
.CD
.DE

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

.DS R
.RD
.DE

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

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

.KS
.KE

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

.KF
.KE

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

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

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

`.B1`
`.B2`

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

4.6.5.8. Tables, figures, equations, and references

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

`.TS [H]`
`.TE`

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

`.PS`
`.PE`

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

`.EQ [`*`align`*`]`
`.EN`

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

`.[`
`.]`

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

4.6.5.9. An example multi-page table

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

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

4.6.5.10. Footnotes

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

`\*[*]`
: Specifies the location of a numbered footnote marker in the text.

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

You can control how `ms` prints footnote numbers by changing the value of the `FF` register. See Document control settings.

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

4.6.6. Page layout

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

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

4.6.6.1. Headers and footers

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

`\*[LH]`
`\*[CH]`
`\*[RH]`
: Sets the left, center, and right headers.

`\*[LF]`
`\*[CF]`
`\*[RF]`
Sets the left, center, and right footers.

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

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

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

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

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

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

4.6.6.2. Margins

You control margins using a set of registers. See Document control settings, for details.

4.6.6.3. Multiple columns

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

`.1C`
Single-column mode.

`.2C`
Two-column mode.

`.MC [width [gutter]]`
Multi-column mode. If you specify no arguments, it is equivalent to the `2C` macro. Otherwise, *width* is the width of each column and *gutter* is the space between columns. The`MINGW` register controls the def ault gutter width.

4.6.6.4. Creating a table of contents

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

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

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

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

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

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

`.TC [no]`

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

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

`.PX [no]`
Prints the table of contents on a new page, using the current page numbering sequence. Use this macro to print a manually generated table of contents at the beginning of your document.

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

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

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

4.6.6.5. Strings and Special Characters

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

`\*[REFERENCES]`
Contains the string printed at the beginning of the references (bibliography) page. The default is '`References`'.

`\*[ABSTRACT]`
Contains the string printed at the beginning of the abstract. The default is '`ABSTRACT`'.

`\*[TOC]`
Contains the string printed at the beginning of the table of contents.

`\*[MONTH1]`
`\*[MONTH2]`
`\*[MONTH3]`
`\*[MONTH4]`
`\*[MONTH5]`
`\*[MONTH6]`
`\*[MONTH7]`
`\*[MONTH8]`
`\*[MONTH9]`
`\*[MONTH10]`
`\*[MONTH11]`
`\*[MONTH12]`
Prints the full name of the month in dates. The default is '`January`', '`February`', etc.

The following special characters are available.[12]

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

`\*[Q]`
`\*[U]`
Prints typographer's quotes where available, and neutral quotes otherwise. `\*Q` is the left quote and `\*U` is the right quote.

[12] For an explanation of what special characters are Special Characters.

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

`.AM`

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

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

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

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

`\*[']`

Acute accent.

`\*[‘]`

Grave accent.

`\*[^]`

Circumflex.

`\*[,]`

Cedilla.

`\*[~]`

Tilde.

`\*[:]`

Umlaut.

`\*[v]`

Hacek.

`\*[_]`

Macron (overbar).

`\*[.]`

Underdot.

`\*[o]`

Ring above.

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

`\*[?]`

Upside-down question mark.

`\*[!]`

Upside-down exclamation point.

`\*[8]`

German ß ligature.

`\*[3]`

Yogh.

`\*[Th]`

Uppercase thorn.

`\*[th]`

Lowercase thorn.

`\*[D-]`
: Uppercase eth.

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

`\*[q]`
: Hooked o.

`\*[ae]`
: Lowercase æ ligature.

`\*[Ae]`
: Uppercase Æ ligature.

4.6.7. Differences from AT&T `ms`

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

- The internals of `groff ms` differ from the internals of AT&T '`troff -ms`'. Documents that depend upon implementation details of AT&T `troff ms` may not format properly with `groff ms`.
- The general error-handling policy of `groff ms` is to detect and report errors, rather than silently to ignore them.
- `groff ms` does not work in compatibility mode (that is, with the `-C` option).
- There is no special support for terminal devices.
- `groff ms` does not provide cut marks.
- Multiple line spacing is not supported. Use a larger vertical spacing instead.
- Some Unix `ms` documentation says that the `CW` and `GW` registers can be used to control the column width and gutter width, respectively. These registers are not used in `groff ms`.
- Macros that cause a reset (paragraphs, headings, etc.) may change the indentation. Macros that change the indentation do not increment or decrement the indentation, but rather set it absolutely. This can cause problems for documents that define additional macros of their own. The solution is to use not the`in` request but instead the `RS` and `RE` macros.
- To make `groff ms` use the default page offset (which also specifies the left margin), the `PO` register must stay undefined until the first `-ms` macro is evaluated. This implies that `PO` should not be used early in the document, unless it is changed also: accessing an undefined register automatically defines it.
- Displays are left-adjusted by default, not indented. In AT&T `troff ms`, '`.DS`' is synonymous with '`.DS I`'; in `groff ms`, it is synonymous with '`.DS L`'.
- Right-adjusted displays are available. The AT&T `troff ms` manual observes that "it is tempting to assume that '`.DS R`' will right adjust lines, but it doesn't work".[13] In `groff ms`, it does.

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

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

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

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

`.TM` Technical memorandum; a cover sheet style

`.IM` Internal memorandum; a cover sheet style

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

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

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

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

`.OK` Other keywords

`.CS` Cover sheet information

`.MH` Murray Hill Bell Laboratories postal address

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

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

`.AM`
: Use improved accent marks. See Strings and Special Characters, for details. This is a Berkeley extension.

`.CW`
: Set text in a `constant-width` (monospaced) typeface. This is a Version 10 Research Unix extension.

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

The following additional registers appear in `groff ms`.

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

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

4.6.8. ms Naming Conventions

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

Internally the macros are divided into modules. The naming conventions are as follows.

- Names used only within one module are of the form *module*`*`*name*.
- Names used outside the module in which they are defined are of the form *module*`@`*name*.
- Names associated with a particular environment are of the form *environment*`:`*name*; these are used only within the `par` module.
- *name* does not have a module prefix.
- Constructed names used to implement arrays are of the form *array*`!`*index*.

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

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

5. gtroff Reference

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

5.1. Text

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

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

5.1.1. Filling

When GNU troff starts up, it obtains information about the device for which it is preparing output.[14] A crucial example is the length of the output line, such as "6.5 inches".

GNU troff reads its input character by character, collecting words as it goes, and fits as many words together on one 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. Words are separated by spaces, tabs, newlines, or file boundaries.[15]

```
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.
```

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.[16] GNU troff follows the example of AT&T troff, attempting to detect the boundaries between sentences, and supplying additional inter-sentence space between

[14] Device and Font Files.

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

[16] 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.

them.

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

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

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

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

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

"Barnum" doesn't begin a sentence! What to do? Let us meet our first*escape sequence*, a series of input characters that give special instructions to GNU `troff` instead of being copied as-is to output device glyphs.[17] An escape sequence begins with the backslash character \ by default, an uncommon character in natural language text, and is*alw ays* followed by at least one other character, hence the term "sequence".

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

```
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 the escape after '`T.`' but we do after '`P.`'. Ensuring that potential sentence boundaries are robust to editing activities and reliably understood both by GNU `troff` and the document author is a goal of the advice presented in Input Conventions.

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

[17] 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 g reat deal of processing before getting to the output.

at the keyboard, or which have special meaning to GNU `troff` (like \ itself).[18]

```
\[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 Using Symbols. To change the additional inter-sentence spacing amount—even to remove it entirely—see the `ss` request in Manipulating Filling and Adjustment.

5.1.3. Hyphenation

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

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

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

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

The remedy for these cases is to tell GNU `troff` where the line may be broken without hyphens. This is done with the non-printing break point escape sequence ‘`\:`’; see Manipulating Hyphenation.

[18] 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.

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

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

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

5.1.5. Adjustment

Once GNU `troff` has filled a line and broken it, it inserts additional inter-sentence space. If the break was automatic, it then tries to*adjust* the line: inter-w ord spaces are widened until the text reaches the right margin. Extra spaces between words are preserved, but trailing spaces on an input line are ignored. Leading spaces are handled as noted above. Text can be adjusted to the left or right margins only (instead of both), or centered; see Manipulating Filling and Adjustment. As a rule, an output line that has not been filled will not be adjusted.

5.1.6. Tab Stops

GNU `troff` translates horizontal tab characters, also called simply "tabs", in the input into movements to the next tab stop. These tab stops are by default located every half inch across the page. With them, simple tables can be made easily.[19] However, this method can be deceptive as the appearance (and width) of the text on a terminal and the results from GNU `troff` can vary greatly, particularly when proportional typefaces are used.

A further possible difficulty is that lines beginning with tab characters are still filled, possibly producing unexpected results.[20]

```
1       2       3
        4       5
```

The above example produces the following output.

```
1       2       3                       4       5
```

GNU `troff` provides sufficient facilities for sophisticated table composition; Tabs and Fields. There are many details to track when using such low-level features, so most users turn to the*tb l*(1) preprocessor for table construction.

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

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

5.1.7. Requests and Macros

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

A *request* is an instruction to the formatter that occurs after a control character. A *control character* must occur at the beginning of an input line to be recognized.[21] The regular control character has a counterpart, the *no-break control char acter*, which suppresses the break that is implied by some requests. The default control characters are the dot (`.`) and the neutral apostrophe (`'`), the latter being the no-break control character. These characters were chosen because it is uncommon for lines of text in natural languages to begin with periods or apostrophes.

Lines beginning with a control character are called *control lines*.[22] Every line of input that is not a control line is a *te xt line*.

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` requests, combined with its 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.[23]

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, it is replaced with what it stands for. The process of replacing a macro call is known as *interpolation*.[24] 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.[25]

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

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

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

In fact, the ending marker can itself be the name of a macro that will be called if it is defined at the time the macro definition begins.

```
.de END
Big Rip
..
```

---

[21] Occasionally a control character is expected as part of another request, such as `if` or `while`.

[22] or a continuation of one (see Line Control)

[23] Argument handling in macros is more flexible but also more complex. See Request and Macro Arguments.

[24] Some escape sequences undergo interpolation as well.

[25] GNU `troff` offers additional ones. See Writing Macros.

```
.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?

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 lines beginning with the control character. Three macros were defined, with a `de` request declaring the macro’s name, and the “body” of the macro starting on the next line and continuing until a line with two dots ‘`..`’ marked its end. The text proper began only after the macros were defined; this is a common pattern. Only the`NOTICE` macro w as 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 Strings) 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`. "F orward references" like this are acceptable because the body of a macro definition is not (completely) interpreted, but stored instead (see Copy Mode). While a macro is being defined, requests are not interpreted and macros not interpolated, whereas some commonly used escape sequences*are* interpolated. `roff` systems also support recursive macros—as long as you have a way to break the recursion (see Conditionals and Loops). For maintainable `roff` documents, arrange your macro definitions so that they are most easily understood when read from beginning to end.

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 into a standard location and suitable for use by a general audience, it is often termed a*macro pac kage*.[26] Macro packages can be loaded by supplying the `-m` option to `groff` or `troff`. Alternatively, a `groff` document wishing to use a macro package can load it with the `mso` ("macro source") request.

5.1.9. Input Encodings

The `groff` front end calls the `preconv` preprocessor to handle most input character encoding issues without troubling the user. Direct input to GNU`troff`, on the other hand, m ust 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 start-up.

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

Any document that is encoded in ISO 646:1991 (a descendant of USAS X3.4-1968 or "US-ASCII"), or, equivalently, uses only code points from the "C0 Controls" and "Basic Latin" parts of the Unicode character set is also a valid ISO Latin-1 document; the standards are interchangeable in their first 128 code points.[27]

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

[27] 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.

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 input encoding characters may not be available for a particular output device. For terminal devices, fallbacks are defined, like '`EUR`' for the Euro sign and '`(C)`' for the copyright sign. For typesetter devices it usually suffices to install fonts that have compatible metrics with other fonts used in the document and contain the necessary glyphs.

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

The `-Tutf8` device supports characters from all other input encodings. `-Tdvi` has support for both Latin-2 and Latin-9 if the command-line `-mec` is used also to load the file `ec.tmac` (which flips to the EC fonts).

5.1.10. Input Conventions

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

- Break input lines after sentence-ending punctuation to ease their recognition (see Sentences). It is frequently convenient to break after colons and semicolons as well, as these typically precede independent clauses. Consider breaking after commas; they often occur in lists that become easy to scan when itemized by line, or constitute supplements to the sentence that are added, deleted, or updated to clarify it. Parenthetical and quoted phrases are also good candidates for placement on input lines by themselves.
- Set your text editor's line length to 72 characters or fewer.[28] This limit, combined with the previous advice regarding breaking around punctuation, makes it less common that an input line will wrap in your text editor, and thus will help you perceive excessively long constructions in your text. Recall that natural languages originate in speech, not writing, and that punctuation is correlated with pauses for breathing and changes in prosody.

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

- Use \& after '!', '?', and '.' if they are followed by space, tab, or newline characters and don't end a sentence.
- Do not attempt to format the input in a WYSIWYG manner (i.e., don't try using spaces to get proper indentation or align columns of a table).
- Comment your document. It is never too soon to apply comments to record information of use to future document maintainers (including your future self). We thus introduce another escape sequence, \", which causes GNU troff to ignore the remainder of the input line.
- Use the empty request—a control character followed immediately by a newline—to visually manage separation of material in input files. Thegroff 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 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 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 58 columns. We have also used an arrow → to indicate a tab character.

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

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

5.2. Measurements

`gtroff` (like many other programs) requires numeric parameters to specify various measurements. Most numeric parameters[29] may have a*measurement unit* attached. These units are specified as a single character that immediately follows the number or expression. Each of these units are understood, by `gtroff`, to be a multiple of its*basic unit*. So , whenever a different measurement unit is specified `gtroff` converts this into its*basic units*. This basic unit, represented by a '`u`', is a device dependent measurement, which is quite small, ranging from 1/75 th to 1/72000 th of an inch. The values may be given as fractional numbers; however, fractional basic units are always rounded to integers.

Some of the measurement units are independent of any of the current settings (e.g., type size) of GNU `troff`.

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

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

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

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

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

`s`
`z` See Fractional Type Sizes, for a discussion of these units.

`f` Fractions. Value is 65536. See Colors, for usage.

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

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

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

`v` Vertical space. This is equivalent to the current line spacing. See Sizes.

`M` 100ths of an em.

5.2.1. Default Units

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

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

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

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

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

5.3. Expressions

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

- Arithmetic: '+' (addition), '−' (subtraction), '/' (division), '*' (multiplication), '%' (modulo). `gtroff` only provides integer arithmetic. The internal type used for computing results is '`int`', which is usually a 32-bit signed integer.
- Comparison: '<' (less than), '>' (greater than), '<=' (less than or equal), '>=' (greater than or equal), '=' (equal), '==' (the same as '=').
- Logical: '`&`' (logical and), '`:`' (logical or).
- Unary operators: '−' (negating, i.e., changing the sign), '+' (just for completeness; does nothing in expressions), '`!`' (logical not; this works only within `if` and `while` requests).[30] See below for the use of unary operators in motion requests.

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

  Example:

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

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

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

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

The register `z` now contains 5.

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

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

For many requests that cause a motion on the page, the unary operators '+' and '−' work differently if leading an expression. They then indicate a motion relative to the current position (down or up, respectively).

Similarly, a leading '`|`' operator indicates an absolute position. For vertical movements, it specifies the distance from the top of the page; for horizontal movements, it gives the distance from the beginning of the *input* line .

'+' and '−' are also treated differently by the following requests and escapes: `bp`, `in`, `ll`, `lt`, `nm`, `nr`, `pl`, `pn`, `po`, `ps`, `pvs`, `rt`, `ti`, `\H`, `\R`, and `\s`. Here, leading plus and minus signs indicate increments and decrements.

See Setting Registers, for some examples.

`\B'`*anything*`'`
: Return 1 if *anything* is a valid numeric expression; or 0 if *anything* is empty or not a valid numeric expression.

Due to the way arguments are parsed, spaces are not allowed in expressions, unless the entire expression is surrounded by parentheses.

See Request and Macro Arguments, and Conditionals and Loops.

5.4. Identifiers

Like any other language, GNU `troff` has rules for properly formed *identifiers*—labels for objects with syntactical importance, like registers, names (macros, strings, diversions, or boxes), environments, fonts, styles, and glyphs. In GNU`troff`, an identifier is a sequence of one or more characters with the following exceptions.

- Spaces, tabs, or newlines.
- Invalid input characters; these are certain 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 diagnostic of type '`input`' (see Debugging). 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`.

  Some of these code points are used by GNU `troff` internally, making it non-trivial to extend the program to cover Unicode or other character encodings that use characters from these ranges.[31]

  Invalid characters are removed during parsing; an identifier `foo`, followed by an

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

invalid character, followed by `bar` is treated as `foobar`.

For example, any of the following identifiers is valid.

```
br
PP
(l
end-list
@_
```

An identifier longer than two characters with a closing bracket (‘`]`’) in its name can’t be accessed with 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]'`’ really asks for glyph ‘`foo]`’.

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

`\A'`*ident*`'`

Test whether an identifier*ident* is v alid in `gtroff`. It expands to the character 1 or 0 according to whether its argument (usually delimited by quotes) is or is not acceptable as the name of a string, macro, diversion, register, environment, or font. It returns 0 if no argument is given. This is useful for looking up user input in some sort of associative table.

```
\A'end-list'
    ⇒ 1
```

See Escapes, for details on parameter delimiting characters.

Identifiers in `gtroff` can be any length, but, in some contexts, `gtroff` needs to be told where identifiers end and text begins (and in different ways depending on their length):

- Single character.
- Two characters. Must be prefixed with ‘`(`’ in some situations.
- Arbitrary length (`gtroff` only). Must be bracketed with ‘`[`’ and ‘`]`’ in some situations. Any length identifier can be put in brackets.

Unlike many other programming languages, undefined identifiers are silently ignored or expanded to nothing. When `gtroff` finds an undefined identifier, it emits a warning, doing the following:

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

See Warnings, Interpolating Registers, and Strings.

Identifiers for macros, strings, diversions (and boxes) share a name space.

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

```
.br
.di
.
.xxx
    ⇒ bar
```

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

See Interpolating Registers, and Strings.

5.5. Embedded Commands

To support the needs of documents that require more than filling, automatic line breaking and hyphenation, and adjustment, GNU `troff` allows commands to be embedded into the text. This is done in two ways.

One is a*request* that tak es up an entire line, and often performs some large-scale operation such as breaking a lines or starting a new page.

The other is an*escape* that can usually be embedded an ywhere in the text; most requests can accept an escape even as an argument. Escapes typically implement relatively minor operations like sub- and superscripting or interpolating a symbol.

5.5.1. Requests

A request line begins with a control character, which is either a single quote ('’', the*no-break control character*) or a period ('.', the normal*control char acter*). These can be changed; see Character Translations, for details. After this there may be optional tabs or spaces followed by an identifier, which is the name of the request. This may be followed by any number of space-separated arguments (*no* tabs here).

Since spaces and tabs are ignored after a control character, it is common practice to use them to structure the source of documents or macro files.

```
.de foo
.  tm This is foo.
..
.
.
.de bar
.  tm This is bar.
..
```

Another possibility is to use the blank line macro request `blm` by assigning an empty macro to it.

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

.de foo
.  tm This is foo.
```

```
..


.de bar
.  tm This is bar.
..

.blm              \" deactivate blank line macro
```

See Blank Line Traps.

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

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

`\n[.br]`

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

This allows reliable modification of requests.

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

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

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

5.5.1.1. Request and Macro Arguments

Arguments to requests and macros are separated by space characters.[32] Only one space between arguments is necessary; additional ones are harmless and ignored.

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

Consider calls to a hypothetical macro `uh`:

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

The first line is the `uh` macro being called with three arguments, '`The`', '`Mouse`', and '`Problem`'. The remainder call the`uh` macro with one argument, '`The Mouse Problem`'. The last

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

solution, using escaped spaces, is “classical” in the sense that it can be found in documents prepared for AT&T `troff`. Nevertheless, it is not optimal in most situations, since ‘\ ’ inserts a fixed-width, non-breaking space character that can’t be adjusted. GNU `troff` provides a different command `\~` to insert a adjustable, non-breaking space.[33]

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

For example,

```
.xxx a" "b c" "de"fg"
```

has the arguments ‘`a"`’, ‘`b c`’, ‘`de`’, and ‘`fg"`’. Don’t rely on this obscure behaviour!

There are two possibilities to get a double quote reliably.

- Enclose the whole argument with double quotes and use two consecutive double quotes to represent a single one. This traditional solution has the disadvantage that double quotes don’t survive argument expansion again if called in compatibility mode (using the `-C` option of `groff`):

  ```
  .de xx
  .  tm xx: ‘\\$1’ ‘\\$2’ ‘\\$3’
  .
  .  yy "\\$1" "\\$2" "\\$3"
  ..
  .de yy
  .  tm yy: ‘\\$1’ ‘\\$2’ ‘\\$3’
  ..
  .xx A "test with ""quotes""" .
      ⇒ xx: ‘A’ ‘test with "quotes"’ ‘.’
      ⇒ yy: ‘A’ ‘test with ’ ‘quotes""’
  ```

  If not in compatibility mode, you get the expected result

  ```
  xx: ‘A’ ‘test with "quotes"’ ‘.’
  yy: ‘A’ ‘test with "quotes"’ ‘.’
  ```

  since `gtroff` preserves the input level.

- Use the double-quote glyph `\(dq`. This works with and without compatibility mode enabled since GNU `troff` doesn’t convert `\(dq` back to a double-quote input character. This method won’t work with AT&T `troff` since it doesn’t define the ‘dq‘ special character.

Double quotes in the `ds` request are handled differently. See Strings, for more details.

5.5.2. Escapes

Escapes may occur anywhere in the input to `gtroff`. They usually begin with a backslash and are followed by a single character, which indicates the function to be performed. The escape character can be changed; see Character Translations.

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

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

- The next single character is the identifier.
- If this single character is an opening parenthesis, take the following two characters as the identifier. There is no closing parenthesis after the identifier.
- If this single character is an opening bracket, take all characters until a closing bracket as the identifier.

Examples:

```
\fB
\n(XX
\*[TeX]
```

Other escapes may require several arguments and/or some special format. In such cases the argument is traditionally enclosed in single quotes (and quotes are always used in this manual for the definitions of escape sequences). The enclosed text is then processed according to what that escape expects. Example:

```
\l'1.5i\(bu'
```

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

```
A caf
\o
e\'


in Paris
  ⇒ A café in Paris
```

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

The following escape sequences (which are handled similarly to characters since they don't take a parameter) are also allowed as delimiters: `\%`, `\ `, `\|`, `\^`, `\{`, `\}`, `\'`, `` \` ``, `\-`, `\_`, `\!`, `\?`, `\)`, `\/`, `\,`, `\&`, `\:`, `\~`, `\0`, `\a`, `\c`, `\d`, `\e`, `\E`, `\p`, `\r`, `\t`, and `\u`. Again, don't use these if possible.

No newline characters as delimiters are allowed in the following escapes: `\A`, `\B`, `\Z`, `\C`, and `\w`.

Finally, the escapes `\D`, `\h`, `\H`, `\l`, `\L`, `\N`, `\R`, `\s`, `\S`, `\v`, and `\x` can't use the following characters as delimiters:

- The digits `0-9`.
- The (single-character) operators '`+-/*%<>=&:().`'.
- The space, tab, and newline characters.
- All escape sequences except `\%`, `\:`, `\{`, `\}`, `\'`, `` \` ``, `\-`, `\_`, `\!`, `\/`, `\c`, `\e`, and `\p`.

To have a backslash (actually, the current escape character) appear in the output several escapes are defined: `\\`, `\e` or `\E`. These are very similar, and only differ with respect to being used in macros or diversions. See Character Translations, for an exact description of those escapes.

See Implementation Differences, Copy Mode, Diversions, and Identifiers.

5.5.2.1. Comments

One of the most common forms of escape is the comment.[34]

`\"`
: Start a comment. Everything to the end of the input line is ignored.

  This may sound simple, but it can be tricky to keep the comments from interfering with the appearance of the final output.

  If the escape is to the right of some text or a request, that portion of the line is ignored, but the space leading up to it is noticed by `gtroff`. This only affects the `ds` and `as` request and its variants.

  One possibly irritating idiosyncrasy is that tabs 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 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
```

  produces

```
Test

Test
```

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

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

`\#`
: To avoid all this, `gtroff` has a new comment mechanism using the `\#` escape. This escape works the same as `\"` except that the newline is also ignored:

```
Test
\# comment
Test
```

  produces

```
Test Test
```

  as expected.

`.ig` [*end*]
: Ignore all input until `gtroff` encounters the macro named *.end* on a line by itself (or `..` if *end* is not specified). This is useful for commenting out large blocks of text:

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

```
text text text...
.ig
This is part of a large block
of text that has been
temporarily(?) commented out.

We can restore it simply by removing
the .ig request and the ".." at the
end of the block.
..
More text text text...
```

produces

```
text text text...  More text text text...
```

The commented-out block of text does not cause a break.

The input is read in copy-mode; auto-incremented registers*are* aff ected (see Auto-increment).

5.6. Registers

Numeric variables in GNU `troff` are called *registers*. There are a number of built-in registers, supplying anything from the date to details of formatting parameters.

See Identifiers, for details on register identifiers.

5.6.1. Setting Registers

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

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

`.nr` *ident value*
`\R'`*ident value*`'`

Set register*ident* to *v alue*. If*ident* doesn't e xist, GNU `troff` creates it.

The argument to `\R` usually has to be enclosed in quotes. See Escapes, for details on parameter delimiting characters.

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

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

For example, the following two lines are equivalent:

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

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

```
.ll 1.6i
```

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

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

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

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

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

`.nr` *ident +value*
`.nr` *ident -value*
`\R'`*ident +value*`'`
`\R'`*ident -value*`'`

Increment (decrement) register *ident* by *value*.

```
.nr a 1
.nr a +1
\na
    ⇒ 2
```

To assign the negated value of a register to another register, some care must be taken to get the desired result:

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

The surrounding parentheses prevent the interpretation of the minus sign as a decrementing operator. An alternative is to start the assignment with a '`0`':

```
.nr a 7
.nr b -3
```

```
.nr a \nb
\na
    ⇒ 4
.nr a 0\nb
\na
    ⇒ -3
```

`.rr` *ident*
: Remove 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*
: Rename register *ident1* to *ident2*. If either *ident1* or *ident2* doesn't exist, the request is ignored.

`.aln` *new old*
: 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 of type '`reg`' is generated and the request is ignored. See Debugging, for information about 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.

5.6.2. Interpolating Registers

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

`\n`*i*
`\n(`*id*
`\n[`*ident*`]`
: Interpolate register with name *ident* (one-character name *i*, two-character name *id*). This means that the value of the register is expanded in-place while `gtroff` is parsing the input line. Nested assignments (also called indirect assignments) are possible.

```
.nr a 5
.nr as \na+\na
\n(as
    ⇒ 10

.nr a1 5
.nr ab 6
.ds str b
.ds num 1
\n[a\n[num]]
    ⇒ 5
\n[a\*[str]]
    ⇒ 6
```

5.6.3. Auto-increment

Number registers can also be auto-incremented and auto-decremented. The increment or decrement value can be specified with a third argument to the `nr` request or `\R` escape.

`.nr` *ident value incr*
: Set register *ident* to *value*; the increment for auto-incrementing is set to *incr*. The `\R` escape doesn't support this notation.

To activate auto-incrementing, the escape `\n` has a special syntax form.

`\n+`*i*
`\n-`*i*
`\n+(`*id*
`\n-(`*id*
`\n+[`*ident*`]`
`\n-[`*ident*`]`
: Before interpolating, increment or decrement *ident* (one-character name *i*, two-character name *id*) by the auto-increment value as specified with the `nr` request (or the `\R` escape). If no auto-increment value has been specified, these syntax forms are identical to `\n`.

For example,

```
.nr a 0 1
.nr xx 0 5
.nr foo 0 -2
\n+a, \n+a, \n+a, \n+a, \n+a
.br
\n-(xx, \n-(xx, \n-(xx, \n-(xx, \n-(xx
.br
\n+[foo], \n+[foo], \n+[foo], \n+[foo], \n+[foo]
```

produces

```
1, 2, 3, 4, 5
-5, -10, -15, -20, -25
-2, -4, -6, -8, -10
```

To change the increment value without changing the value of a register (*a* in the example), the following can be used:

```
.nr a \na 10
```

5.6.4. Assigning Formats

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

`.af` *ident format*
: Change the output format of a register. The first argument *ident* is the name of the register to be changed, and the second argument *format* is the output format. The following output formats are available:

`1` Decimal arabic numbers. This is the default format: 0, 1, 2, 3, ...

`0...0` Decimal numbers with as many digits as specified. So, '`00`' would result in printing numbers as 01, 02, 03, ...

In fact, any digit instead of zero does work; `gtroff` only counts how many digits are specified. As a consequence, `af`'s default format '`1`' could be specified as '`0`' also (and exactly this is returned by the `\g` escape, see below).

`I` Upper-case Roman numerals: 0, I, II, III, IV, ...

`i` Lower-case Roman numerals: 0, i, ii, iii, iv, ...

`A` Upper-case letters: 0, A, B, C, ..., Z, AA, AB, ...

`a` Lower-case letters: 0, a, b, c, ..., z, aa, ab, ...

Omitting the register format causes a warning of type '`missing`'. See Debugging, for more details. Specifying a nonexistent format causes an error.

The following example produces '`10, X, j, 010`':

```
.nr a 10
.af a 1           \" the default format
\na,
.af a I
\na,
.af a a
\na,
.af a 001
\na
```

The largest number representable for the '`i`' and '`I`' formats is 39999 (or −39999); Unix `troff` uses '`z`' and '`w`' to represent 10000 and 5000 in Roman numerals, and so does `gtroff`. Currently, the correct glyphs of Roman numeral five thousand and Roman numeral ten thousand (Unicode code points `U+2182` and `U+2181`, respectively) are not available.

If *ident* doesn't exist, it is created.

Changing the output format of a read-only register causes an error. It is necessary to first copy the register's value to a writable register, then apply the `af` request to this other register.

\g*i*
\g(*id*
\g[*ident*]

Return the current format of the specified register*ident* (one-char acter name*i*, tw o-character name*id*). F or example, '`\ga`' after the previous example would produce the string '`000`'. If the register hasn't been defined yet, nothing is returned.

5.6.5. Built-in Registers

The following lists some built-in registers that are not described elsewhere in this manual. Any register that begins with a '.' is read-only. A complete listing of all built-in registers can be found in Register Index.

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

`\n[.H]` Number of basic units per horizontal unit of output device resolution. See Measurements.

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

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

`\n[.V]` Number of basic units per vertical unit of output device resolution. See Measurements.

`\n[seconds]`
The number of seconds after the minute, normally in the range 0 to 59, but can be up to 61 to allow for leap seconds. Initialized at start-up of `gtroff`.

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

`\n[hours]`
The number of hours past midnight, in the range 0 to 23. Initialized at start-up of `gtroff`.

`\n[dw]` Day of the week (1–7).

`\n[dy]` Day of the month (1–31).

`\n[mo]` Current month (1–12).

`\n[year]` The current year.

`\n[yr]` The current year minus 1900. Unfortunately, the documentation of Unix Version 7's `troff` had a year 2000 bug: It incorrectly claimed that `yr` contains the last two digits of the year. That claim has never been true of either AT&T `troff` or GNU `troff`. Old`troff` input that looks lik e this:

```
'\" The following line stopped working after 1999
This document was formatted in 19\n(yr.
```

can be corrected as follows:

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

or, to be portable to older `troff` versions, as follows:

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

`\n[.c]`
`\n[c.]` The current*input* line n umber. Register '`.c`' is read-only, whereas '`c.`' (a `gtroff` extension) is writable also, affecting both '`.c`' and '`c.`'.

`\n[ln]` The current*output* line n umber after a call to the `nm` request to activate line numbering.

See Miscellaneous, for more information about line numbering.

`\n[.x]` The major version number. For example, if the version number is 1.03 then `.x` contains '`1`'.

`\n[.y]` The minor version number. For example, if the version number is 1.03 then `.y` contains '`03`'.

`\n[.Y]` The revision number of `groff`.

`\n[$$]` The process ID of `gtroff`.

`\n[.g]` Always 1. Macros should use this to determine whether they are running under GNU `troff`.

`\n[.A]` If the command-line option `-a` is used to produce an ASCII approximation of the output, this is set to 1, zero otherwise. See Options.

`\n[.O]` This read-only register is set to the suppression nesting level (see escapes `\O`). See Suppressing output.

`\n[.P]` This register is set to 1 (and to 0 otherwise) if the current page is actually being printed, i.e., if the `-o` option is being used to only print selected pages. See Options, for more information.

`\n[.T]` If `gtroff` is called with the `-T` command-line option, the register `.T` is set to 1, and zero otherwise. See Options.

5.7. Manipulating Filling and Adjustment

Various ways of causing*breaks* w ere shown in Breaking. The`br` request lik ewise causes a break. Several other requests also cause breaks implicitly. These are`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.

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. If no output line is pending, it is as if a break has already happened; additional breaks, whether explicit or implicit, have no effect.

`.br`
Break the current line: any input collected on the pending output line is emitted without adjustment.

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

```
⇒ foo bar
⇒ baz qux
```

Initially, 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`
`\n[.u]`
: Activate fill mode; a pending output line is broken. The read-only register `.u` is set to 1.

  Fill mode enablement status is associated with the current environment (see Environments).

  See Line Control, for interaction with the `\c` escape.

`.nf`
: Activate no-fill mode: the output line length (see Line Layout) is ignored and output lines are broken where the input lines are. A pending output line is broken and adjustment is suppressed. The register `.u` is set to 0.

  Fill mode enablement status is associated with the current environment (see Environments).

  See Line Control, for interaction with the `\c` escape.

`.ad` [*mode*]
`\n[.j]`
: Set adjustment mode to *mode*, taking eff ect when the pending (or next) output line is broken. Adjustment is suppressed in no-fill mode.

  mode can have one of the following values.

  `b`
  `n`
  : Adjust "normally": to both margins. This is the GNU `troff` def ault.

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

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

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

  Finally, *mode* can be the numeric argument returned by the `.j` register.

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

```
.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
```

produces the following output:

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

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

The adjustment mode and enablement status are associated with the current environment (see Environments).

The value of `.j` for any adjustment mode is an implementation detail and should not be relied upon as a programmer's interface.

`.na`

Disable adjustment. This produces the same output as adjustment 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 current environment (see Environments).

`.brp`
`\p`

Break, adjusting the line per the current adjustment mode.

With `\p`, this break will happen at the next word boundary. The`\p` itself is remo ved entirely, adding neither a break nor a space where it appears in input; it can thus be placed in the middle of a word to cause a break at the end of that word.

In most cases this produces 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.
```

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

Center the next *nnn* input text lines without filling them. A pending output line is broken. The number of lines still to be centered is associated with the current environment (see Environments).

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

The previous example 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 pattern is useful for any request that takes a count of input lines as an argument.

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

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

Align the next *nnn* input text lines to the right margin without filling them. A pending output line is broken. The `.rj` read-only register is the n umber of lines to be right-justified as set by the `rj` request. The number of lines still to be right-justified is associated with the current environment (see Environments).

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

Set the sizes of spaces between words and sentences.[35] Their units are twelfths of the space width parameter 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 automatically adjusted, it may increase.

The optional second argument sets the amount of additional space separating sentences on the same output line in fill mode. If the second argument is omitted,*additional-sentence-space-size* 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 current environment (see Environments), and rounded down to the nearest multiple of 12 on terminal output devices.

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

A related application of the `ss` request is to insert discardable horizontal space; i.e., space that is discarded at a line break. For example, some footnote styles collect the notes into a single paragraph with large spaces between each.

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

The result has obvious inter-sentence spacing.

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

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

5.8. Manipulating Hyphenation

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

[35] See Filling and Sentences for the definitions of word and sentence boundaries, respectively.

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

`.hw` *word ...*

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

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

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

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

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

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

The request is ignored if there are no parameters.

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

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

`\%`
`\:`

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

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

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

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

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

`.hc` [*char*]
: Change the hyphenation character to*char*. This char acter then works as the `\%` escape normally does, and thus no longer appears in the output.[36] Without an argument, `hc` resets the hyphenation character to `\%` (the default).

  The hyphenation character is associated with the current environment (see Environments).

`.shc` [*glyph*]
: Set the*soft h yphen character* to *glyph*.[37] If the argument is omitted, the soft hyphen character is set to the default, `\[hy]`. The*soft h yphen character* is the glyph that is inserted when a word is automatically hyphenated at a line break.[38] If the soft hyphen character does not exist in the font of the character immediately preceding a potential break point, then the line is not broken at that point. Neither character definitions (specified with the `char` and similar requests) nor translations (specified with the `tr` request) are considered when assigning the soft hyphen character.

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

`.hy` [*mode*]
`\n[.hy]`
: Set hyphenation mode to*mode*. The optional n umeric argument*mode* encodes conditions for hyphenation.

  Typesetting practice generally does not avail itself of every opportunity for hyphenation, but the details differ by language and site mandates. The hyphenation modes of AT&T `troff` were implemented with English-language publishing practices of the 1970s in mind, not a scrupulous enumeration of conceivable parameters. GNU `troff` extends those modes such that finer-grained control is possible, 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.[39] The entries in the table below are termed *values*, and the sum of the desired values is the*mode*.

  `0` disables hyphenation.

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

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

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

[39] 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.

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

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

`2` disables hyphenation of the last word on a page.[40]

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

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

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

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

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

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

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

[40] This value prevents hyphenation if the next page location trap is closer than the next text baseline would be. GNU`troff` automatically inserts an implicit v ertical position trap at the end of each page to cause a page transition. This value can be used in traps planted by users or macro packages to prevent hyphenation of the last word in a column in multi-column page layouts or before floating figures or tables. See Page Location Traps.

default!). Modes 4 and 6 are typical.

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

language	**pattern name**	**left min**	**right min**
Czech	cs	2	2
U.S. English	us	2	3
French	fr	2	3
German traditional	det	2	2
German reformed	den	2	2
Swedish	sv	1	2

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

The hyphenation mode is associated with the current environment (see Environments).

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

`.nh`

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

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

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

The *pattern-file* should have the same format as (simple) TEX pattern files. More specifically, the following scanning rules are implemented.

- A percent sign starts a comment (up to the end of the line) even if preceded by a backslash.
- "Digraphs" like `\$` are not supported.
- `^^xx` (where each *x* is 0–9 or a–f) and `^^c` (character *c* in the code point range 0–127 decimal) are recognized; other uses of `^` cause an error.
- No macro expansion is performed.
- `hpf` checks for the expression `\patterns{...}` (possibly with whitespace before or after the braces). Everything between the braces is taken as hyphenation patterns. Consequently, `{` and `}` are not allowed in patterns.
- Similarly, `\hyphenation{...}` gives a list of hyphenation exceptions.
- `\endinput` is recognized also.
- For backwards compatibility, if `\patterns` is missing, the whole file is treated as a list of hyphenation patterns (except that the `%` character is recognized as the start of a comment).

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

The `hpfcode` request defines mapping values for character codes in pattern files. It is an older mechanism no longer used by GNU `troff`'s own macro files; for its successor, see `hcode` below. `hpf` or `hpfa` apply the mapping after reading the patterns but before replacing or appending to the active list of patterns. Its arguments are pairs of character codes—integers from 0 to 255. The request maps character code*a* to code *b*, code*c* to code *d*, and so on. Character codes that would otherwise be invalid in GNU `troff` can be used. By default, every code maps to itself except those for letters 'A' to 'Z', which map to those for 'a' to 'z'.

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

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

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

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

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

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

For hyphenation to work, hyphenation codes must be set up. At start-up, 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 e xtends this principle to letters outside the Unicode basic Latin alphabet; without it, words containing such letters won't be hyphenated properly even if the corresponding hyphenation patterns contain them. For example, the following `hcode` requests are necessary to assign hyphenation codes to the letters 'ÄäÖöÜüß' (needed for German):

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

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

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

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

The hyphenation language is associated with the current environment (see Environments).

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

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

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

Set the maximum number of consecutive hyphenated lines to *n*. If *n* is negativ e, there is no maximum. If omitted, *n* is −1. This v alue is associated with the current environment (see Environments). Only lines output from a given environment count towards the maximum associated with that environment. Hyphens resulting from `\%` are counted; explicit hyphens are not.

The `.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*]
`\n[.hym]`

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 h yphenation margin is reset to its default value, 0. The default scaling indicator is '`m`'. The hyphenation margin is associated with the current environment (see Environments).

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

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

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

Suppress hyphenation of the line in adjustment modes '`b`' or '`n`' if it can be justified by adding no more than *hyphenation-space* extra space to each inter-word space. Without an argument, the hyphenation space adjustment threshold is set to its default value, 0. The default scaling indicator is '`m`'. The hyphenation space adjustment threshold is associated with the current environment (see Environments).

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

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

5.9. Manipulating Spacing

`.sp` [*distance*]

Space downwards *distance*. With no argument it advances 1 line. A negative argument causes `gtroff` to move up the page the specified distance. If the argument is preceded by a '`|`' then`gtroff` mo ves that distance from the top of the page. This request causes a line break, and that adds the current line spacing to the space you have just specified. The default scaling indicator is '`v`'.

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

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

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

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

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

results in

```
foo


bar

baz
```

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

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

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

`.ls` [*nnn*]
`\n[.L]`

Output *nnn−1* blank lines after each line of text. With no argument,`gtroff` uses the previous value before the last `ls` call.

```
.ls 2    \" This causes double-spaced output
.ls 3    \" This causes triple-spaced output
.ls      \" Again double-spaced
```

The line spacing is associated with the current environment (see Environments).

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

See Changing Type Sizes, for the requests `vs` and `pvs` as alternatives to `ls`.

`\x'`*spacing*`'`
`\n[.a]`

Sometimes, extra vertical spacing is only needed occasionally, e.g. to allow space for a tall construct (like an equation). The `\x` escape does this. The escape is given a numerical argument, usually enclosed in quotes (like '`\x'3p'`'); the default scaling indicator is '`v`'. If this number is positive extra vertical space is inserted below the current line. A negative number adds space above. If this escape is used multiple times on the same line, the maximum of the values is used.

See Escapes, for details on parameter delimiting characters.

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

Using `\x` can be necessary in combination with the `\b` escape, as the following example shows.

```
This is a test with the \[rs]b escape.
.br
This is a test with the \[rs]b escape.
.br
This is a test with \b'xyz'\x'-1m'\x'1m'.
.br
This is a test with the \[rs]b escape.
.br
This is a test with the \[rs]b escape.
```

produces

```
This is a test with the \b escape.
This is a test with the \b escape.
                     x
This is a test with y.
                     z
This is a test with the \b escape.
This is a test with the \b escape.
```

`.ns`
`.rs`
`\n[.ns]`

Enable *no-space mode*. In this mode, spacing (either via `sp` or via blank lines) is

disabled. The `bp` request to adv ance to the next page is also disabled, except if it is accompanied by a page number (see Page Control). 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).

5.10. Tabs and Fields

A tab character (ASCII char 9, EBCDIC char 5) causes a horizontal movement to the next tab stop (much like it did on a typewriter).

`\t`

This escape is a non-interpreted tab character. In copy mode (see Copy Mode), `\t` is the same as a real tab character.

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

Change tab stop positions. This request takes a series of tab specifiers as arguments (optionally divided into two groups with the letter '`T`') that indicate where each tab stop is to be (overriding any previous settings).

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

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

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

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

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

```
.ta T 1i
```

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

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

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

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

Some notes:

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

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

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

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

  `gtroff` first converts the tab stops of the line into unbreakable horizontal movements, then splits the line after the second 'b' (assuming a sufficiently short line length). Usually, this isn't what the user wants.

- Superfluous tabs (i.e., tab characters that do not correspond to a tab stop) are ignored except the first one, which delimits the characters belonging to the last tab stop for right-justifying or centering. Consider the following example

```
.ds Z   foo\tbar\tfoo
.ds ZZ  foo\tbar\tfoobar
.ds ZZZ foo\tbar\tfoo\tbar
.ta 2i 4iR
\*[Z]
.br
\*[ZZ]
.br
\*[ZZZ]
.br
```

  which produces the following output:

```
foo                 bar                 foo
foo                 bar              foobar
foo                 bar                 foobar
```

  The first line right-justifies the second 'foo' relative to the tab stop. The second line right-justifies 'foobar'. The third line finally right-justifies only 'foo' because of the additional tab character, which marks the end of the string belonging to the last defined tab stop.

- Tab stops are associated with the current environment (see Environments).
- Calling `ta` without an argument removes all tab stops.
- The start-up value of `gtroff` is 'T 0.5i'.

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

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

⇒ T120u

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

.tc [*fill-glyph*]

Normally, GNU troff writes no glyph when moving to a tab stop (some output devices may explicitly output space characters to achieve this motion). 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 occup y the interval from the current horizontal location to the next tab stop. With no argument, GNU troff re verts to the default behavior. The tab repetition character is associated with the current environment (see Environments).[41] Only a single *fill-glyph* is recogniz ed; any excess is ignored.

.linetabs *n*
\n[.linetabs]

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

For example, the following code:

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

in normal mode, results in the output

```
a       b       c
```

in line-tabs mode, the same code outputs

```
a       b                       c
```

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

5.10.1. Leaders

Sometimes it may be desirable to use the tc request to fill a particular tab stop with a given glyph (for example dots in a table of contents), but also normal tab stops on the rest of the line. For this GNU troff provides an alternate tab mechanism, called *leaders*, which does just that.[42]

A leader character (character code 1) behaves similarly to a tab character: It moves to the next tab stop. The only difference is that for this movement, the fill glyph defaults to a period character and not to space.

[41] Tab repetition *char acter* 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`
: This escape is a non-interpreted leader character. In copy mode (see Copy Mode), `\a` is the same as a real leader character.

`.lc` [*fill-glyph*]
: 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 occup y 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 current environment (see Environments).[43] Only a single *fill-glyph* is recognized; any excess is ignored.

For a table of contents, to name an example, tab stops may be defined so that the section number is one tab stop, the title is the second with the remaining space being filled with a line of dots, and then the page number slightly separated from the dots.

```
.ds entry 1.1\tFoo\a\t12
.lc .
.ta 1i 5i +.25i
\*[entry]
```

This produces

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

5.10.2. Fields

Fields are a more general way of laying out tabular data. A field is defined as the data between a pair of *delimiting char acters*. It contains substrings that are separated by *padding characters*. The width of a field is the distance on the *input* line from the position where the field starts to the next tab stop. A padding character inserts stretchable space similar to TEX's `\hss` command (thus it can even be negative) to make the sum of all substring lengths plus the stretchable space equal to the field width. If more than one padding character is inserted, the available space is evenly distributed among them.

`.fc` [*delim-char* [*padding-char*]]
: Define a delimiting and a padding character for fields. If the latter is missing, the padding character defaults to a space character. If there is no argument at all, the field mechanism is disabled (which is the default). In contrast to, e.g., the tab repetition character, delimiting and padding characters are *not* associated with the current environment (see Environments).

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

[43] Leader repetition *char acter* is a misnomer since it is an output glyph.

5.11. Character Translations

The control character (‘.’) and the no-break control character (‘’’) can be changed with the `cc` and `c2` requests, respectively.

`.cc` [*c*]
: Set the control character to *c*. With no argument the def ault control character ‘.’ is restored. The value of the control character is associated with the current environment (see Environments).

`.c2` [*c*]
: Set the no-break control character to *c*. With no argument the def ault control character ‘’’ is restored. The value of the no-break control character is associated with the current environment (see Environments).

See Requests.

`.eo`
: Disable the escape mechanism completely. After executing this request, the backslash character ‘\’ no longer starts an escape sequence.

  This request can be very helpful in writing macros since it is not necessary then to double the escape character. Here an example:

```
.\" This is a simplified version of the
.\" .BR request from the man macro package
.eo
.de BR
.  ds result \&
.  while (\n[.$] >= 2) \{\
.    as result \fB\$1\fR\$2
.    shift 2
.  \}
.  if \n[.$] .as result \fB\$1
\*[result]
.  ft R
..
.ec
```

`.ec` [*c*]
: Set the escape character to *c*. With no argument the def ault escape character ‘\’ is restored. It can be also used to re-enable the escape mechanism after an `eo` request.

  Changing the escape character globally likely breaks macro packages, since GNU `troff` has no mechanism to ‘intern’ macros, i.e., to convert a macro definition into an internal form that is independent of its representation (TEX has such a mechanism). If a macro is called, it is executed literally.

`.ecs`
`.ecr`
: The `ecs` request saves the current escape character in an internal register. Use this request in combination with the `ec` request to temporarily change the escape character.

  The `ecr` request restores the escape character saved with `ecs`. Without a previous

call to `ecs`, this request sets the escape character to `\`.

`\\`
`\e`
`\E`

Print the current escape character (which is the backslash character '\' by default).

`\\` is a 'delayed' backslash; more precisely, it is the default escape character followed by a backslash, which no longer has special meaning due to the leading escape character. It is*not* an escape sequence in the usual sense! In an y unknown escape sequence `\`*X* the escape character is ignored and *X* is printed. But if *X* is equal to the current escape character, no warning is emitted.

As a consequence, only at the top level or in a diversion is a backslash glyph printed; in copy mode, it expands to a single backslash, which then combines with the following character to form an escape sequence.

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

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

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

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

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

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

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

`\.`

Similar to `\\`, the sequence `\.` isn't a real escape sequence. As before, a warning message is suppressed if the escape character is followed by a dot, and the dot itself is printed.

```
.de foo
```

```
.  nop foo
.
.  de bar
.    nop bar
\\..
.
..
.foo
.bar
    ⇒ foo bar
```

The first backslash is consumed while the macro is read, and the second is swallowed while executing macro `foo`.

A *translation* is a mapping of an input character to an output glyph. The mapping occurs at output time, i.e., the input character gets assigned the metric information of the mapped output character right before input tokens are converted to nodes (see `gtroff` Internals, for more on this process).

`.tr` *abcd...*
`.trin` *abcd...*

Translate character *a* to glyph *b*, character *c* to glyph *d*, etc. If there is an odd number of arguments, the last one is translated to an unstretchable space (‘\ ’).

The `trin` request is identical to `tr`, but when you unformat a diversion with `asciify` it ignores the translation. See Diversions, for details about the `asciify` request.

Some notes:

- Special characters (`\(`*xx*, `\[`*xxx*`]`, `\C'`*xxx*`'`, `\'`, `` \` ``, `\-`, `\_`), glyphs defined with the `char` request, and numbered glyphs (`\N'`*xxx*`'`) can be translated also.
- The `\e` escape can be translated also.
- Characters can be mapped onto the `\%` and `\~` escapes (but `\%` and `\~` can’t be mapped onto another glyph).
- The following characters can’t be translated: space (with one exception, see below), backspace, newline, leader (and `\a`), tab (and `\t`).
- Translations are not considered for finding the soft hyphen character set with the `shc` request.
- The pair ‘*c*`\&`’ (this is an arbitrary character *c* followed by the non-printing input break) maps this character to nothing.

  ```
  .tr a\&
  foo bar
      ⇒ foo br
  ```

  It is even possible to map the space character to nothing:

  ```
  .tr aa \&
  foo bar
      ⇒ foobar
  ```

  As shown in the example, the space character can’t be the first character/glyph pair as an argument of `tr`. Additionally, it is not possible to map

the space character to any other glyph; requests like '`.tr aa x`' undo '`.tr aa \&`' instead.

If justification is active, lines are justified in spite of the 'empty' space character (but there is no minimal distance, i.e. the space character, between words).

- After an output glyph has been constructed (this happens at the moment immediately before the glyph is appended to an output glyph list, either by direct output, in a macro, diversion, or string), it is no longer affected by `tr`.
- Translating character to glyphs where one of them or both are undefined is possible also; `tr` does not check whether the entities in its argument do exist. See `gtroff` Internals.
- `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...*

`trnt` is the same as the `tr` request except that the translations do not apply to text that is transparently throughput into a diversion with `\!`. See Diversions.

For example,

```
.tr ab
.di x
\!.tm a
.di
.x
```

prints '`b`' to the standard error stream; if`trnt` is used instead of `tr` it prints '`a`'.

5.12. `troff` and `nroff` 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 e xecutable[44] that

[44] A GNU `nroff` program is available for convenience; it calls GNU `troff` to perform the formatting.

sends its output to a device driver (`grotty` for terminal devices, `grops` for PostScript, etc.) 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`.

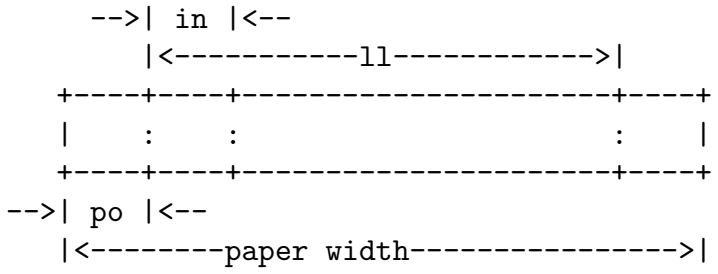
`.troff`
: Make the 't' built-in condition true (and the 'n' built-in condition false) for `if`, `ie`, and `while` conditional requests. This is the default if GNU `troff` (*not* `groff`) is started with the `-R` switch to avoid loading of the start-up files `troffrc` and `troffrc-end`. Without `-R`, GNU `troff` stays in `troff` mode if the output device is not a terminal (e.g., 'ps').

`.nroff`
: 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 start-up file `troffrc`.

See Conditionals and Loops, for more details on built-in conditions.

5.13. Line Layout

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



These dimensions are:

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

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

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

A simple demonstration:

```
.ll 3i
This is text without indentation.
The line length has been set to 3\~inch.
.in +.5i
```

```
.ll -.5i
Now the left and right margins are both increased.
.in
.ll
Calling .in and .ll without parameters restore
the previous values.
```

Result:

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

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

Set horizontal page offset to*offset* (or increment or decrement the current v alue by *offset*). This request does not cause a break, so changing the page offset in the middle of text being filled may not yield the expected result. The initial value is 1 i. For terminal output devices, it is set to 0 in the startup file `troffrc`; the default scaling indicator is 'm'. This request is incorrectly documented in the AT&T `troff` manual as using a default scaling indicator of 'v'.

The current page offset can be found in the read-only register '`.o`'.

If `po` is called without an argument, the page offset is reset to the previous value before the last call to `po`.

```
.po 3i
\n[.o]
    ⇒ 720
.po -1i
\n[.o]
    ⇒ 480
.po
\n[.o]
    ⇒ 720
```

`.in` [*indent*]
`.in` +*indent*
`.in` -*indent*
`\n[.i]`

Set indentation to*indent* (or increment or decrement the current v alue by *indent*). This request causes a break. Initially, there is no indentation.

If `in` is called without an argument, the indentation is reset to the previous value before the last call to `in`. The default scaling indicator is 'm'.

The indentation is associated with the current environment (see Environments).

If a negative indentation value is specified (which is not allowed), `gtroff` emits a warning of type '`range`' and sets the indentation to zero.

The effect of `in` is delayed until a partially collected line (if it exists) is output. A temporary indentation value is reset to zero also.

The current indentation (as set by `in`) can be found in the read-only register '`.i`'.

`.ti` *offset*
`.ti` +*offset*
`.ti` -*offset*
`\n[.in]`

Temporarily indent the next output line by *offset*. If an increment or decrement value is specified, adjust the temporary indentation relative to the value set by the `in` request.

This request causes a break; its value is associated with the current environment (see Environments). The default scaling indicator is '`m`'. A call of`ti` without an argument is ignored.

If the total indentation value is negative (which is not allowed), `gtroff` emits a warning of type '`range`' and sets the temporary indentation to zero. 'Total indentation' is either *offset* if specified as an absolute value, or the temporary plus normal indentation, if*offset* is giv en 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*]
`.ll` +*length*
`.ll` -*length*
`\n[.l]`
`\n[.ll]`

Set the line length to*length* (or increment or decrement the current v alue by *length*). Initially, the line length is set to 6.5 i. The effect of `ll` is delayed until a partially collected line (if it exists) is output. The default scaling indicator is '`m`'.

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

The line length is associated with the current environment (see Environments).

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.

5.14. Line Control

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

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

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

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

produces

```
This is a   test.

This is a             test.
```

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

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

In the above example, '`|0`' specifies a negative distance from the current position (at the end of the just emitted argument `\$1`) back to the beginning of the input line. Thus, the '`\l`' escape draws a line from right to left.

`gtroff` makes a difference between input and output line continuation; the latter is also called *interrupting* a line.

`\RET`,
`\c`,
`\n[.int]`

Continue a line. `\RET` (this is a backslash at the end of a line immediately followed by a newline) works on the input level, suppressing the effects of the following newline in the input.

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

The '`|`' operator is also affected.

`\c` works on the output level. Anything after this escape on the same line is ignored except `\R`, which works as usual. Anything before `\c` on the same line is appended to the current partial output line. The next non-command line after an interrupted line counts as a new input line.

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

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

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

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

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

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

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

5.15. Page Layout

GNU `troff` provides some primitive operations for controlling page layout.

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

Set the *page length* to *length* (or increment or decrement the current value by *length*). This is the length of the physical output page. The default scaling indicator is 'v'.

The current setting can be found in the read-only 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 Traps, 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*'

Print a *title line*. It consists of three parts: a left-justified portion, a centered portion, and a right-justified portion. The argument separator '`'`' can be replaced with any character not occurring in the title line. The '`%`' character is replaced with the current page number. This character can be changed with the `pc` request (see below).

Without argument, `tl` is ignored.

Some notes:

- The line length set by the `ll` request is not honoured by `tl`; use the `lt` request (described below) instead, to control line length for text set by `tl`.
- A title line is not restricted to the top or bottom of a page.
- `tl` prints the title line immediately, ignoring a partially filled line (which stays untouched).
- It is not an error to omit closing delimiters. For example, '`.tl /foo`' is equivalent to '`.tl /foo//`': It prints a title line with the left-justified word '`foo`'; the centered and right-justified parts are empty.

- `tl` accepts the same parameter delimiting characters as the `\A` escape; see Escapes.

`.lt` [*length*]
`.lt` +*length*
`.lt` -*length*
`\n[.lt]`
The title line is printed using its own line length, which is specified (or incremented or decremented) with the `lt` request. Initially, the title line length is set to 6.5 i. If a negative line length is specified (which is not allowed), `gtroff` emits a warning of type 'range' and sets the title line length to zero. The default scaling indicator is 'm'. If `lt` is called without an argument, the title length is reset to the previous value before the last call to `lt`.

The current setting of this is available in the `.lt` read-only register; it is associated with the current environment (see Environments).

`.pn` *page*
`.pn` +*page*
`.pn` -*page*
`\n[.pn]`
Change (increase or decrease) the page number of the*ne xt* 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*]
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 Traps.

5.16. Page Control

`.bp` [*page*]
`.bp` +*page*
`.bp` -*page*
`\n[%]`
Stop processing the current page and move to the next page. This request causes a break. It can also take an argument to set (increase, decrease) the page number of the next page (which becomes the current page after `bp` has finished). The difference between `bp` and `pn` is that `pn` does not cause a break or actually eject a page. See Page Layout.

```
.de newpage                              \" define macro
'bp                                      \" begin page
'sp .5i                                  \" vertical space
.tl 'left top'center top'right top'      \" title
'sp .3i                                  \" vertical space
..                                       \" end macro
```

`bp` has no effect if not called within the top-level diversion (see Diversions).

The writable register % holds the current page number.

The register .pe is set to 1 while bp is active. See Page Location Traps.

.ne [*space*]

It is often necessary to force a certain amount of space before a new page occurs. This is most useful to make sure that there is not a single*orphan* line left at the bottom of a page. Thene request ensures that there is a certain distance , specified by the first argument, before the next page is triggered (see Traps). The default scaling indicator for ne is 'v'; the default value of*space* is 1 v if no argument is giv en.

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

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

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

.sv [*space*]

.os

sv is similar to the ne request; it reserves the specified amount of vertical space. If the desired amount of space exists before the next trap (or the bottom page boundary if no trap is set), the space is output immediately (ignoring a partially filled line, which stays untouched). If there is not enough space, it is stored for later output via the os request. The default value is 1 v if no argument is given; the default scaling indicator is 'v'.

Both sv and os ignore no-space mode. While thesv request allo ws negative values for *space*, os ignores them.

\n[nl]

This register contains the current vertical position. If the vertical position is zero and the top of page transition hasn't happened yet, nl is set to negative value. gtroff itself does this at the very beginning of a document before anything has been printed, but the main usage is to plant a header trap on a page if this page has already started.

Consider the following:

```
.de xxx
.  sp
.  tl'Header'
.  sp
..
.
First page.
.bp
.wh 0 xxx
.nr nl (-1)
Second page.
```

Result:

```
First page.


...
```

```
                                    Header

Second page.

...
```

Without resetting `nl` to a negative value, the just planted trap would be active beginning with the*ne xt* page, not the current one.

See Diversions, for a comparison with the `.h` and `.d` registers.

5.17. Fonts and Symbols

`gtroff` can switch fonts at any point in the text.

The basic set of fonts is '`R`', '`I`', '`B`', and '`BI`'. These are Times roman, italic, bold, and bold-italic. For non-terminal devices, there is also at least one symbol font that contains various special symbols (Greek, mathematics).

5.17.1. Changing Fonts

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

The `ft` request and the `\f` escape change the current font to *font* (one-character name *f*, two-character name*fn*).

If *font* is a style name (as set with the `sty` request or with the `styles` command in the `DESC` file), use it within the current font family (as set with the `fam` request, 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` s witches to the previous font. Use `\f[]` to do this with the escape. The old syntax forms `\fP` or `\f[P]` are also supported.

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

The example below produces two identical lines.

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

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

\f doesn't produce an input token in GNU troff. As a consequence, it can be used in requests like mc (which expects a single character as an argument) to change the font on the fly:

```
.mc \f[I]x\f[]
```

The current style name is available in the read-only register '.sty' (this is a string-valued register); if the current font isn't a style, the empty string is returned. It is associated with the current environment.

See Font Positions, for an alternative syntax.

.ftr *f* [*g*]

Translate font *f* to font *g*. Whenever a font named*f* is ref erred to in a \f escape sequence, in the F and S conditional operators, or in the ft, ul, bd, cs, tkf, special, fspecial, fp, or sty requests, font *g* is used. If *g* is missing or equal to *f* the translation is undone.

Font translations cannot be chained.

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

.fzoom *f* [*zoom*]

\n[.zoom]

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

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

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

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.

5.17.2. Font Families

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

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

This way, it is possible to use the basic four fonts and to select a different font family on the command line (see Options).

`.fam` [*family*]
`\n[.fam]`
`\F`*f*
`\F(`*fm*
`\F[`*family*`]`
`\n[.fn]`

Switch font family to *family* (one-character name*f*, tw o-character name*fm*). If no argument is given, switch back to the previous font family. Use`\F[]` to do this with the escape; `\FP` selects font family '`P`' instead.

The value at start-up is '`T`'. The current font family is available in the read-only register '`.fam`' (this is a string-valued register); it is associated with the current environment.

```
spam,
.fam H    \" helvetica family
spam,     \" used font is family H + style R = HR
.ft B     \" family H + style B = font HB
spam,
.fam T    \" times family
spam,     \" used font is family T + style B = TB
.ft AR    \" font AR (not a style)
baked beans,
.ft R     \" family T + style R = font TR
and spam.
```

`\F` doesn't produce an input token in GNU `troff`. As a consequence, it can be used in requests like `mc` (which expects a single character as an argument) to change the font family on the fly.

```
.mc \F[P]x\F[]
```

The '`.fn`' register contains the current*real f ont name* of the current font. This is a string-valued register. If the current font is a style, the value of `\n[.fn]` is the proper concatenation of family and style name.

`.sty` *n style*

Associate *style* with font position*n*. A f ont 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 Options). The `styles` command in the `DESC` file controls which font positions (if any) are initially associated with styles rather than fonts. For example, the default setting for PostScript fonts

```
styles R I B BI
```

is equivalent to

```
.sty 1 R
.sty 2 I
.sty 3 B
.sty 4 BI
```

`fam` and `\F` always check whether the current font position is valid; this can give surprising results if the current font position is associated with a style.

In the following example, we want to access the PostScript font `FooBar` from the font family `Foo`:

```
.sty \n[.fp] Bar
.fam Foo
    ⇒ warning: can't find font 'FooR'
```

The default font position at start-up is 1; for the PostScript device, this is associated with style '`R`', so `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 Font Positions.

5.17.3. Font Positions

For compatibility with AT&T `troff`, GNU `troff` has the concept of font *positions* at which various fonts are *mounted*.

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

Mount font *font* at position *pos* (which must be a non-negative integer). This numeric position can then be referred to with font-changing commands. When GNU `troff` starts, it uses font position 1 (which must exist; position 0 is unused at start-up.[45])

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

```
.nr save-font \n[.f]
```

[45] Usually.

```
.ft B
... text text text ...
.ft \n[save-font]
```

The number of the next free font position is available in the read-only register '.fp'. This is useful when mounting a new font, like so:

```
.fp \n[.fp] NEATOFONT
```

Fonts not listed in the DESC file are automatically mounted on the next available font position when they are referenced. If a font is to be mounted explicitly with the fp request on an unused font position, it should be mounted on the first unused font position, which can be found in the .fp register, although GNU troff does not enforce this strictly.

The fp request has an optional third argument. This argument gives the external name of the font, which is used for finding the font description file. The second argument gives the internal name of the font, which is used to refer to the font in gtroff after it has been mounted. If there is no third argument then the internal name is used as the external name. This feature makes it possible to use fonts with long names in compatibility mode.

Both the ft request and the \f escape have alternative syntax forms to access font positions.

.ft *nnn*
\f*n*
\f(*nn*
\f[*nnn*]

Change the current font position to*nnn* (one-digit position *n*, tw o-digit position*nn*), which must be a non-negative integer.

If *nnn* is associated with a style (as set with the sty request or with the styles command in the DESC file), use it within the current font family (as set with the fam request, the \F escape, or the family command in the DESC file).

```
this is font 1
.ft 2
this is font 2
.ft                    \" switch back to font 1
.ft 3
this is font 3
.ft
this is font 1 again
```

See Changing Fonts, for the standard syntax form.

5.17.4. Using Symbols

A *glyph* is a graphical representation of a*char acter*. 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 Device and Font Files, and Special Fonts, for more details.

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

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

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

must be called directly:

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

Glyph names not listed in *groff_char*(7) are derived algorithmically, using a simplified version of the Adobe Glyph List (AGL) algorithm, which is described in https://github.com/adobe-type-tools/agl-aglfn. The (frozen) set of glyph names that can't be derived algorithmically is called 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*
\[*name*]
\[*component1 component2 ...*]

Insert a symbol*name* (tw o-character name*nm*) or a composite glyph with component glyphs*component1*, *component2*, ... There is no special syntax f or one-character names—the natural form '\n' would collide with escapes.[47]

If *name* is undefined, a warning of type 'char' is generated, and the escape is ignored. See Debugging, for information about warnings.

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

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

[47] A one-character symbol is not the same as an input character, i.e., the character a is not the same as \[a]. By default, groff defines only a single one-character symbol, \[-]; it is usually accessed as \-. On the other hand, GNU troff has the special feature that \[char*XXX*] is the same as the input character with character code *XXX*. For example, \[char97] is identical to the letter a if ASCII encoding is active.

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

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

Examples:

\[A ho] 'A' maps tou0041, 'ho' maps tou02DB, 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 tou0045_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 agroff f ont file. This is not really a limitation; instead, you have to define u0041_0328.

\C'*xxx*'
: Typeset the glyph named *xxx*.[48] 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*
: Map glyph name*from* to glyph name *to* if it is used in \[...] with more than one component. See above for examples.

  This mapping is based on glyph names only; no check for the existence of either glyph is done.

  A set of default mappings for many accents can be found in the file composite.tmac, which is loaded at start-up.

\N'*n*'
: Typeset the glyph with code*n* in the current f ont (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 con veniently used in conjunction with the char request:

  ```
  .char \[phone] \f[ZD]\N'37'
  ```

  The code of each glyph is given in the fourth column in the font description file after

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

the `charset` command. It is possible to include unnamed glyphs in the font description file by using a name of '—'; the `\N` escape sequence is the only way to use these.

No kerning is applied to glyphs accessed with `\N`.

Some escape sequences directly map onto special glyphs.

`\'`

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

`\‘`

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

`\-`

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

`\_`

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

`.cflags` *n c1 c2 ...*

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.[49] 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 char acter class defined with the `class` request rather than an individual character. See Character Classes.

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

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

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

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

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

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

[49] 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.

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

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

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

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

into your document. Note, however, that this can lead to bad layout if done without thinking; in most situations, a better solution instead of changing the `cflags` value is to insert `\:` right after the hyphen at the places that really need a break point.

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

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

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

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

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

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

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

is applied to this object rather than to individual glyphs in *string*.

An object defined by these requests can be used just like a normal glyph provided by the output device. In particular, other characters can be translated to it with the `tr` or `trin` requests; it can be made the leader character with the `lc` request; repeated patterns can be drawn with it using the `\l` and `\L` escape sequences; and words containing *g* can be hyphenated correctly if the `hcode` request is used to give the object a hyphenation code.

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

The `tr` and `trin` requests take precedence if `char` accesses the same symbol.

```
.tr XY
X
    ⇒ Y
.char X Z
X
    ⇒ Y
.tr XX
X
    ⇒ Z
```

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

`fschar` defines a fallback glyph for font *f*: `gtroff` checks for glyphs defined with `fschar` after the list of fonts declared as font-specific special fonts with the `fspecial` request, but before the list of fonts declared as global special fonts with the `special` request.

Finally, the `schar` request defines a global fallback glyph: `gtroff` checks for glyphs defined with `schar` after the list of fonts declared as global special fonts with the `special` request, but before the already mounted special fonts.

See Character Classes.

`.rchar` *c1 c2 ...*
`.rfschar` *f c1 c2 ...*

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

Spaces and tabs are optional between *cn* arguments.

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

See Special Characters.

5.17.5. Character Classes

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

`.class` *name c1 c2 ...*

Define a character class (or simply "class")*name* comprising the char acters *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 char acter (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 Using Symbols, for more details.

5.17.6. Special Fonts

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

`gtroff` provides the following two requests to add more special fonts. See Using Symbols, for a detailed description of the glyph searching mechanism in `gtroff`.

Usually, only non-TTY devices have special fonts.

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

Use the `special` request to define special fonts. Initially, this list is empty.

Use the `fspecial` request to designate special fonts only when font *f* is active. Initially, this list is empty.

Previous calls to `special` or `fspecial` are overwritten; without arguments, the particular list of special fonts is set to empty. Special fonts are searched in the order they appear as arguments.

All fonts that appear in a call to `special` or `fspecial` are loaded.

See Using Symbols, for the exact search order of glyphs.

5.17.7. Artificial Fonts

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

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

Change (increment, decrement) the height of the current font, but not the width. If *height* is zero, restore the original height. Default scaling indicator is 'z'.

The read-only register `.height` contains the font height as set by `\H`.

Currently, only the `-Tps` and `-Tpdf` devices support this feature.

`\H` doesn't produce an input token in GNU `troff`. As a consequence, it can be used in requests like `mc` (which expects a single character as an argument) to change the font on the fly:

```
.mc \H'+5z'x\H'0'
```

In compatibility mode, `gtroff` behaves differently: If an increment or decrement is used, it is always taken relative to the current point size and not relative to the previously selected font height. Thus,

```
.cp 1
\H'+5'test \H'+5'test
```

prints the word '`test`' twice with the same font height (five points larger than the current font size).

`\S'`*slant*`'`
`\n[.slant]`

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*]
The `ul` request normally underlines subsequent lines if a TTY output device is used. Otherwise, the lines are printed in italics (only the term 'underlined' is used in the following). The single argument is the number of input lines to be underlined; with no argument, the next line is underlined. If*lines* is z ero or negative, stop the effects of `ul` (if it was active). Requests and empty lines do not count for computing the number of underlined input lines, even if they produce some output like `tl`. Lines inserted by macros (e.g. invoked by a trap) do count.

At the beginning of `ul`, the current font is stored and the underline font is activated. Within the span of a `ul` request, it is possible to change fonts, but after the last line affected by `ul` the saved font is restored.

This number of lines still to be underlined is associated with the current environment (see Environments). The underline font can be changed with the `uf` request.

The `ul` request does not underline spaces.

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

`.uf` *font*
Set the underline font (globally) used by `ul` and `cu`. By default, this is the font at position 2. *font* can be either a non-negative font position or the name of a font.

`.bd` *font* [*offset*]
`.bd` *font1 font2* [*offset*]
`\n[.b]`
Artificially create a bold font by printing each glyph twice, slightly offset.

Two syntax forms are available.

- Imitate a bold font unconditionally. The first argument specifies the font to embolden, and the second is the number of basic units, minus one, by which the two glyphs are offset. If the second argument is missing, emboldening is turned off. *font* can be either a non-negative font position or the name of a font.

  offset is available in the `.b` read-only register if a special font is active; in the `bd` request, its default unit is '`u`'.

- Imitate a bold form conditionally. Embolden*f ont1* 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*]]
Switch to and from*constant glyph space mode*. If activ ated, the width of every glyph is *width*/36 ems. The em size is given absolutely by *em-size*; if this argument is missing, the em value is taken from the current font size (as set with the `ps` request)

when the font is effectively in use. Without second and third argument, constant glyph space mode is deactivated.

Default scaling indicator for *em-size* is 'z'; *width* is an integer.

5.17.8. Ligatures and Kerning

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

Most PostScript fonts support the fi and fl ligatures. The C/A/T typesetter that was the target of AT&T `troff` also supported 'ff', 'ffi', and 'ffl' ligatures. Advanced typesetters or 'expert' fonts may include ligatures for 'ft' and 'ct', although GNU `troff` does not support these (yet).

Only the current font is checked for ligatures and kerns; neither special fonts nor entities defined with the `char` request (and its siblings) are taken into account.

`.lg` [*flag*]
`\n[.lg]`
Switch the ligature mechanism on or off; if the parameter is non-zero or missing, ligatures are enabled, otherwise disabled. Default is on. The current ligature mode can be found in the read-only register `.lg` (set to 1 or 2 if ligatures are enabled, 0 otherwise).

Setting the ligature mode to 2 enables the two-character ligatures (fi, fl, and ff) and disables the three-character ligatures (ffi and ffl).

Pairwise kerning is another subtle typesetting mechanism that modifies the distance between a glyph pair to improve readability. In most cases (but not always) the distance is decreased. For example, compare the combination of the letters 'V' and 'A'. With kerning, 'VA' is printed. Without kerning it appears as 'VA'. Typewriter-like fonts and fonts for terminals where all glyphs have the same width don't use kerning.

`.kern` [*flag*]
`\n[.kern]`
Switch kerning on or off. If the parameter is non-zero or missing, enable pairwise kerning, otherwise disable it. The read-only 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 Font File Format.

Track kerning expands or reduces the space between glyphs. This can be handy, for example, if you need to squeeze a long word onto a single line or spread some text to fill a narrow column. It must be used with great care since it is usually considered bad typography if the reader notices the effect.

`.tkf` *f s1 n1 s2 n2*
Enable track kerning for font *f*. If the current font is*f* the width of e very glyph is increased by an amount between *n1* and *n2* (*n1*, *n2* can be negative); if the current

point size is less than or equal to*s1* the width is increased b y *n1*; if it is greater than or equal to*s2* the width is increased b y *n2*; if the point size is greater than or equal to *s1* and less than or equal to *s2* the increase in width is a linear function of the point size.

The default scaling indicator is 'z' for *s1* and *s2*, 'p' for *n1* and *n2*.

The track kerning amount is added even to the rightmost glyph in a line; for large values it is thus recommended to increase the line length by the same amount to compensate.

Sometimes, when typesetting letters of different fonts, more or less space at such boundaries is needed. There are two escapes to help with this.

`\/`

Increase the width of the preceding glyph so that the spacing between that glyph and the following glyph is correct if the following glyph is a roman glyph. For example, if an italic `f` is immediately followed by a roman right parenthesis, then in many fonts the top right portion of the `f` overlaps the top left of the right parenthesis. Use this escape sequence whenever an italic glyph is immediately followed by a roman glyph without any intervening space. This small amount of space is also called*italic correction*.

```
\f[I]f\f[R])
    ⇒ f)
\f[I]f\/\f[R])
    ⇒ f)
```

`\,`

Modify the spacing of the following glyph so that the spacing between that glyph and the preceding glyph is correct if the preceding glyph is a roman glyph. Use this escape sequence whenever a roman glyph is immediately followed by an italic glyph without any intervening space. In analogy to above, this space could be called*left italic correction*, but this term isn't used widely.

```
q\f[I]f
    ⇒ qf
q\,\f[I]f
    ⇒ qf
```

`\&`

Insert a non-printing input break, which is invisible. Its intended use is to stop interaction of a character with its surroundings.

- It prevents the insertion of extra space after an end-of-sentence character.

```
Test.
Test.
    ⇒ Test.  Test.
Test.\&
Test.
    ⇒ Test. Test.
```

- It prevents interpretation of a control character at the beginning of an input line.

```
.Test
error     warning: macro 'Test' not defined
\&.Test
    ⇒ .Test
```

- It prevents kerning between two glyphs.

```
VA
    ⇒ VA
V\&A
    ⇒ VA
```

- It is needed to map an arbitrary character to nothing in the `tr` request (see Character Translations).

`\)`

This escape is similar to `\&` except that it behaves like a character declared with the `cflags` request to be transparent for the purposes of an end-of-sentence character.

Its main usage is in macro definitions to protect against arguments starting with a control character.

```
.de xxx
\)\\$1
..
.de yyy
\&\\$1
..
This is a test.\c
.xxx '
This is a test.
    ⇒This is a test.'  This is a test.
This is a test.\c
.yyy '
This is a test.
    ⇒This is a test.' This is a test.
```

5.18. Sizes

GNU `troff` uses two dimensions with each line of text, type size and vertical spacing. The *type size* is approximately the height of the tallest glyph.[50] *Vertical spacing* is the amount of space `gtroff` allows for a line of text; normally, this is about 20% larger than the current type size. Ratios smaller than this can result in hard-to-read text; larger than this, it spreads the text out more vertically (useful for term papers). By default, `gtroff` uses 10 point type on 12 point spacing.

Typesetters call the difference between type size and vertical spacing*leading*. [51]

[50] This is usually the parenthesis. In most cases the real dimensions of the glyphs in a font are*not* related to its type size! For example, the standard PostScript font families 'Times', 'Helvetica', and 'Courier' can't be used together at 10 pt; to get acceptable output, the size of 'Helvetica' has to be reduced by one point, and the size of 'Courier' must be increased by one point.

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

5.18.1. Changing Type Sizes

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

Use the `ps` request or the `\s` escape to change (increase, decrease) the type size (in points). Specify*siz e* as either an absolute point size, or as a relative change from the current size. `ps` with no argument restores the previous size.

The default scaling indicator of `size` is 'z'. If the resulting size is non-positive, it is set to 1 u.

The read-only register `.s` returns the point size in points as a decimal fraction. This is a string. To get the point size in scaled points, use the `.ps` register instead (see Fractional Type Sizes).

`.s` is associated with the current environment (see Environments).

```
snap, snap,
.ps +2
grin, grin,
.ps +2
wink, wink, \s+2nudge, nudge,\s+8 say no more!
.ps 10
```

The `\s` escape may be called in a variety of ways. Much like other escapes there must be a way to determine where the argument ends and the text begins. Any of the following forms is valid:

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

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

`\s(`*nn* Set the point size to*nn* points . *nn* must be exactly two digits.

`\s+(`*nn*
`\s-(`*nn*
`\s(+`*nn*
`\s(-`*nn* Increase or decrease the point size by *nn* points. *nn* must be exactly two digits.

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

Note that `\s` doesn't produce an input token in `gtroff`. As a consequence, it can be used in requests like `mc` (which expects a single character as an argument) to change the font on the fly:

```
.mc \s[20]x\s[0]
```

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

Some devices may only have certain permissible sizes, in which case `gtroff` rounds to the nearest permissible size. The`DESC` file specifies which siz es are permissible for the device.

Use the `sizes` request to change the permissible sizes for the current output device. Arguments are in scaled points; the `sizescale` line in the `DESC` file for the output device provides the scaling factor. For example, if the scaling factor is 1000, then the value 12000 is 12 points.

Each argument can be a single point size (such as '`12000`'), or a range of sizes (such as '`4000-72000`'). You can optionally end the list with a zero.

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

Change (increase, decrease) the vertical spacing by *space*. The default scaling indicator is '`p`'.

If `vs` is called without an argument, the vertical spacing is reset to the previous value before the last call to `vs`.

`gtroff` creates a warning of type '`range`' if*space* is negativ e; the vertical spacing is then set to smallest positive value, the vertical resolution (as given in the `.V` register).

'`.vs 0`' isn't saved in a diversion since it doesn't result in a vertical motion. You explicitly have to repeat this command before inserting the diversion.

The read-only register `.v` contains the current vertical spacing; it is associated with the current environment (see Environments).

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

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

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

See Manipulating Spacing, for more details on the `\x` escape and the `ls` request.

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

Change (increase, decrease) the post-vertical spacing by *space*. The default scaling indicator is 'p'.

If pvs is called without an argument, the post-vertical spacing is reset to the previous value before the last call to pvs.

gtroff creates a warning of type 'range' if*space* is z ero or negative; the vertical spacing is then set to zero.

The read-only register .pvs contains the current post-vertical spacing; it is associated with the current environment (see Environments).

5.18.2. Fractional Type Sizes

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

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

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

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

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

\n[.ps]
: A read-only register returning the point size in scaled points.

  .ps is associated with the current environment (see Environments).

\n[.psr]
\n[.sr]
: The last-requested point size in scaled points is contained in the read-only register .psr. The last-requested point size in points as a decimal fraction can be found in the read-only string-valued register .sr.

  The requested point sizes are device-independent, whereas the values returned by the .ps and .s registers are not. For example, if a point size of 11 pt is requested, and a sizes request (or a sizescale line in a DESC file) specifies 10.95 pt instead, this value is actually used.

  Both registers are associated with the current environment (see Environments).

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

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

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

See Device and Font Files.

5.19. Strings

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

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

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

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

The `\*` escape *interpolates* (expands in place) a previously defined string variable *name* (one-character name*n*, tw o-character name*nm*). More precisely , the stored string is pushed onto the input stack, which is then parsed normally. Similarly to registers, it is possible to nest strings; i.e., string variables can be called within string variables. An argument in a string definition must be escaped for correct behavior; See Parameters.

```
.ds a \\$1 wildebeest
.ds b big, \*[a hairy]
I see a \*[b].
    ⇒ I see a big, hairy wildebeest.
```

If the string named by the `\*` escape does not exist, it is defined as empty, and a warning of type '`mac`' is emitted (see Debugging).

If `ds` is called with only one argument, *name* is defined as an empty string.

Caution: Unlike other requests, the second argument to the `ds` request consumes the remainder of the input line, including trailing spaces. This means that comments on a line with such a request can introduce unwanted space into a string when they are set off from the material they annotate, as is conventional.

```
.ds TeX T\h'-.2m'\v'.2m'E\v'-.2m'\h'-.1m'X \" Knuth's TeX
```

Instead, place the comment on another line or put the comment escape immediately adjacent to the last character of the string.

```
.ds TeX T\h'-.2m'\v'.2m'E\v'-.2m'\h'-.1m'X\" Knuth's TeX
```

It is good style to end string definitions (and appendments; see below) with a comment, even an empty one, to prevent unwanted space from creeping into them during source document maintenance.

```
.ds author Alice Pleasance Liddell\"
.ds empty \" might be appended to later with .as
```

To store leading space in a string, start it with a double quote. A double quote is special only in that position; double quotes in any other location are included in the string (the effects of escape sequences notwithstanding).

```
.ds salutation "         Yours in a white wine sauce,\"
.ds c-var-defn "    char build_date[]="2020-07-29";\"
.ds sucmd sudo sh -c "fdisk -l /dev/sda > partitions"\"
```

Strings are not limited to a single line of input text. A string can span several lines by escaping the newlines with a backslash. The resulting string is stored*without* the newlines.

```
.ds foo This string contains \
text on multiple lines \
of input.
```

It is not possible to embed a newline in a string that will be interpreted as such when the string is interpolated. To achieve that effect, use the `\*` escape to interpolate a macro instead.

Strings, macros, diversions (and boxes) share a same name space; Identifiers. Internally, the same mechanism is used to store them. It is thus possible to invoke a macro with string interpolation syntax and vice versa.

```
.de subject
Typesetting
..
.de predicate
rewards attention to detail
..
\*[subject] \*[predicate].
Truly.
    ⇒ Typesetting
    ⇒  rewards attention to detail Truly.
```

What went wrong? Strings don't contain newlines, but macros do. String

interpolation placed a newline at the end of '`\*[subject]`', and the next thing on the input was a space. Similarly, when '`\*[predicate]`' was interpolated, it was followed by the empty request '`.`' on a line by itself. If we want to use macros as strings, we must take interpolation behavior into account.

```
.de subject
Typesetting\\
..
.de predicate
rewards attention to detail\\
..
\*[subject] \*[predicate].
Truly.
    ⇒ Typesetting rewards attention to detail.  Truly.
```

By ending each text line of the macros with an escaped '`\RET`', we get the desired effect (see Line Control). What would have happened if we had used only one backslash at a time instead?

Interpolating a string does not hide existing macro arguments. Thus in a macro, a more efficient way of doing

```
.xx \\$@
```

is

```
\\*[xx]\\
```

The latter calling syntax doesn't change the value of `\$0`, which is then inherited from the calling macro (see Parameters).

Diversions and boxes can be also called with string syntax. It is sometimes convenient to copy one-line diversions or boxes to a string.

```
.di xxx
a \fItest\fR
.br
.di
.ds yyy This is \*[xxx]\c
\*[yyy].
    ⇒ This is a test.
```

As the previous example shows, it is possible to store formatted output in strings. The `\c` escape prevents the subsequent newline from being interpreted as a break (again, see Line Control).

Copying diversions longer than a single output line produces unexpected results.

```
.di xxx
a funny
.br
test
.br
.di
.ds yyy This is \*[xxx]\c
\*[yyy].
```

```
    ⇒ test This is a funny.
```

Usually, it is not predictable whether a diversion contains one or more output lines, so this mechanism should be avoided. With AT&T `troff`, this was the only solution to strip off a final newline from a diversion. Another disadvantage is that the spaces in the copied string are already formatted, making them unstretchable. This can cause ugly results.

A clean solution to this problem is available in GNU `troff`, using the requests `chop` to remove the final newline of a diversion, and `unformat` to make the horizontal spaces stretchable again.

```
.box xxx
a funny
.br
test
.br
.box
.chop xxx
.unformat xxx
This is \*[xxx].
    ⇒ This is a funny test.
```

See gtroff Internals.

The `ds1` request defines a string such that compatibility mode is off when the string is later interpolated. To be more precise, a*compatibility sa ve* input token is inserted at the beginning of the string, and a*compatibility restore* input tok en at the end.

```
.nr xxx 12345
.ds aa The value of xxx is \\n[xxx].
.ds1 bb The value of xxx is \\n[xxx].
.
.cp 1
.
\*(aa
error    warning: number register '[' not defined
    ⇒ The value of xxx is 0xxx].
\*(bb
    ⇒ The value of xxx is 12345.
```

`.as` *name* [*string*]

`.as1` *name* [*string*]

The `as` request is similar to `ds` but appends*string* to the string stored as *name* instead of redefining it. If*name* doesn't e xist yet, it is created. If `as` is called with only one argument, no operation is performed (beyond dereferencing it).

```
.as salutation " with shallots, onions and garlic,\"
```

The `as1` request is similar to `as`, but compatibility mode is switched off when the appended portion of the string is later interpolated. To be more precise, a*compatibility save* input token is inserted at the beginning of the appended string, and a*compatibility restore* input token at the end.

Several requests exist to perform rudimentary string operations. Strings can be queried (`length`) and modified (`chop`, `substring`, `stringup`, `stringdown`), and their names can be manipulated through renaming, removal, and aliasing (`rn`, `rm`, `als`).

`.length` *reg anything*

Compute the number of characters of*an ything* and store the count in the register *reg*. If*reg* doesn't e xist, it is created.*an ything* is read in copy mode.

```
.ds xxx abcd\h'3i'efgh
.length yyy \*[xxx]
\n[yyy]
    ⇒ 14
```

`.chop` *object*

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 `gtroff` Internals, for details on nodes inserted additionally by GNU `troff`.

`.substring` *str start* [*end*]

Replace the string named*str* with its substring bounded b y the indices*start* and *end*, inclusive. The first character in the string has index 0. If*end* is omitted, it is implicitly set to the largest valid value (the string length minus one). Negative indices count backwards from the end of the string: the last character has index −1, the character before the last has index −2, and so on.

```
.ds xxx abcdefgh
.substring xxx 1 -4
\*[xxx]
    ⇒ bcde
.substring xxx 2
\*[xxx]
    ⇒ de
```

`.stringdown` *str*
`.stringup` *str*

Alter the string named*str* b y replacing each of its bytes with its lowercase (`stringdown`) or uppercase (`stringup`) version (if one exists). GNU`troff` special char acters (see the *groff_char*(7) man page) can be used and the output will usually transform in the expected way due to the regular naming convention of the special character escapes.

```
.ds resume R\['e]sum\['e]
\*[resume]
.stringdown resume
\*[resume]
.stringup resume
\*[resume]
    ⇒ Résumé résumé RÉSUMÉ
```

(In pratice, we would end the `ds` request with a comment escape `\"` to prevent space from creeping into the definition during source document maintenance.)

`.rn` *old new*
: Rename the request, macro, diversion, or string *old* to *new*.

`.rm` *name*
: Remove the request, macro, diversion, or string *name*. GNU `troff` treats subsequent invocations as if the name had never been defined.

`.als` *new old*
: Create an alias *new* for the existing request, string, macro, or diversion object named *old*, causing the names to refer to the same stored object. If *old* is undefined, a warning of type '`mac`' is generated and the request is ignored.

  To understand how the `als` request works, consider two different storage pools: one for objects (macros, strings, etc.), and another for names. As soon as an object is defined, GNU `troff` adds it to the object pool, adds its name to the name pool, and creates a link between them. When `als` creates an alias, it adds a new name to the name pool that gets linked to the same object as the old name.

  Now consider this example.

```
.de foo
..
.
.als bar foo
.
.de bar
.  foo
..
.
.bar
error     input stack limit exceeded
error     (probable infinite loop)
```

  In the above, `bar` remains an *alias*—another name for—the object referred to by `foo`, which the second `de` request replaces. Alternatively, imagine that the `de` request *dereferences* its argument before replacing it. Either way, the result of calling `bar` is a recursive loop that finally leads to an error. See Writing Macros.

  To remove an alias, simply call `rm` on its name. The object itself is not destroyed until it has no more names.

5.20. Conditionals and Loops

GNU `troff` has `if` and `while` control structures like other languages. However, the syntax for grouping multiple input lines in the branches or bodies of these structures is unusual.

5.20.1. Operators in Conditionals

In `if`, `ie`, and `while` requests, in addition to ordinary numeric expressions (see Expressions), several boolean operators are available.

`c` *glyph*
: True if a *glyph* is available, where *glyph* is a Unicode basic Latin character, a GNU `troff` special character '`\(`*xx*' or '`\[`*xxx*`]`', '`\N'`*xxx*`'`', or has been defined by the `char` request.

`d` *name* True if there is a string, macro, diversion, or request called *name*.

`e` True if the current page is even-numbered.

`F` *font* True if a font called *font* exists. *font* is handled as if it were opened with the `ft` request (that is, font translation and styles are applied), without actually mounting it.

This test doesn't load the complete font, but only its header to verify its validity.

`m` *color* True if there is a color called *color*.

`n` True if the document is being processed in `nroff` mode (i.e., the `nroff` request has been issued). See Unknown.

`o` True if the current page is odd-numbered.

`r` *reg* True if there is a register called *reg*.

`S` *style* True if a style called *style* has been registered. F ont translation is applied.

`t` True if the document is being processed in `troff` mode (i.e., the `troff` request has been issued). See Unknown.

`v` Always false. This condition is recognized only for compatibility with certain other `troff` implementations.[52]

`'`*xxx*`'`*yyy*`'`

True if the output produced by *xxx* is equal to the output produced by *yyy*. Other characters can be used in place of the single quotes; the same set of delimiters as for the `\D` escape is used (see Escapes). `gtroff` formats *xxx* and *yyy* in separate environments; after the comparison the resulting data is discarded.

```
.ie "|"\fR|\fP" \
true
.el \
false
    ⇒ true
```

The resulting motions, glyph sizes, and fonts have to match,[53] and not the individual motion, size, and font requests. In the previous example, '`|`' and '`\fR|\fP`' both result in a roman '`|`' glyph with the same point size and at the same location on the page, so the strings are equal. If '`.ft I`' had been added before the '`.ie`', the result would be "false" because (the first) '`|`' produces an italic '`|`' rather than a roman one.

To compare strings without processing, surround the data with `\?`.

```
.ie "\?|\?"\?\fR|\fP\?" \
true
.el \
```

[52] 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.

[53] The created output nodes must be identical. See `gtroff` Internals.

```
false
    ⇒ false
```

Since data protected with `\?` is read in copy mode it is even possible to use incomplete input without causing an error.

```
.ds a \[
.ds b \[
.ie '\?\*a\?'\?\*b\?' \
true
.el \
false
    ⇒ true
```

These operators can't be combined with other operators like '`:`' or '`&`'; only a leading '`!`' (without spaces or tabs between the exclamation mark and the operator) can be used to negate the result.

```
.nr x 1
.ie !r x register x is not defined
.el      register x is defined
    ⇒ register x is defined
```

Spaces and tabs immediately after '`!`' cause the condition to evaluate as zero (this bizarre behavior maintains compatibility with AT&T `troff`).

```
.nr x 1
.ie ! r x register x is not defined
.el       register x is defined
    ⇒ r x register x is not defined
```

The unexpected appearance of '`r x`' in the output is a clue that our conditional was not interpreted the way we planned, but matters may not always be so obvious.

Spaces and tabs are optional before the arguments to the '`r`', '`d`', and '`c`' operators.

5.20.2. if-then

`.if` *expr anything*

Evaluate the expression *expr*, and execute *anything* (the remainder of the line) if *expr* evaluates true (that is, to a value greater than zero). *anything* is interpreted as though it were on a line by itself (except that leading spaces are ignored). See Operators in Conditionals.

```
.nr xxx 1
.nr yyy 2
.if ((\n[xxx] == 1) & (\n[yyy] == 2)) true
    ⇒ true
```

`.nop` *anything*

Executes *anything*. This is similar to '`.if 1`'.

5.20.3. if-else

`.ie` *expr anything*
`.el` *anything*

Use the `ie` and `el` requests to write an if-then-else. The first request is the 'if' part and the latter is the 'else' part.

```
.ie n .ls 2 \" double-spacing in nroff
.el   .ls 1 \" single-spacing in troff
```

See Expressions.

5.20.4. Conditional Blocks

`\{`
`\}`

It is frequently desirable for a control structure to govern more than one request, call more than one macro, span more than one input line of text, or mix the foregoing. The opening and closing brace escapes `\{` and `\}` perform such grouping. Brace escapes can be used outside of control structures, but when they are they have no meaning and produce no output.

`\{` should appear (after optional spaces and tabs) immediately subsequent to the request's conditional expression. `\}` should appear on a line with other occurrences of itself as necessary to match `\{` escapes. It can be preceded by a control character, spaces, and tabs. Input after an`\}` escape on the same line is only processed if all the preceding conditions to which the escapes correspond are true. Furthermore, a `\}` closing the body of a `while` request must be the last such escape on an input line.

```
A
.if 0 \{ B
C
D
\}E
F
    ⇒ A F

N
.if 1 \{ O
.  if 0 \{ P
Q
R\} S\} T
U
    ⇒ N O U
```

If the above behavior challenges the intuition, keep in mind that it was implemented to retain compatibility with AT&T `troff`. For clarity, it is common practice to end input lines with `\{`, optionally followed by `\RET` to suppress a break before subsequent text lines, and to have nothing more than a control character, spaces, and tabs before any lines containing `\}`.

```
.de DEBUG
debug =
```

```
.ie \\$1 \{\
ON,
development
\}
.el \{\
OFF,
production
\}
version
..
.DEBUG 0
.br
.DEBUG 1
```

Try omitting the `\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) any spaces or tabs on the same input line are interpreted *as if they were on an input line by themselves*.

5.20.5. while

GNU `troff` provides a looping construct using the `while` request, which is used much like the `if` request.

`.while` *expr anything*

Evaluate the expression *expr*, and repeatedly execute *anything* (the remainder of the line) until *expr* evaluates false.

```
.nr a 0 1
.while (\na < 9) \{\
\n+a,
.\}
\n+a
    ⇒ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
```

Some remarks.

- The body of a `while` request is treated like the body of a `de` request: `gtroff` temporarily stores it in a macro that is deleted after the loop has been exited. It can considerably slow down a macro if the body of the `while` request (within the macro) is large. Each time the macro is executed, the `while` body is parsed and stored again as a temporary macro.

  ```
  .de xxx
  .  nr num 10
  .  while (\\n[num] > 0) \{\
  .    \" many lines of code
  .    nr num -1
  .  \}
  ..
  ```

  The traditional and often better solution (AT&T `troff` lacked the `while` request) is to use a recursive macro instead that is parsed only once during

its definition.

```
.de yyy
.  if (\\n[num] > 0) \{\
.    \" many lines of code
.    nr num -1
.    yyy
.  \}
..
.
.de xxx
.  nr num 10
.  yyy
..
```

The number of available recursion levels is set to 1000 (this is a compile-time constant value of `gtroff`).

- The closing brace of a `while` body must end a line.

```
.if 1 \{\
.  nr a 0 1
.  while (\n[a] < 10) \{\
.    nop \n+[a]
.\}\}
    ⇒ unbalanced \{ \}
```

`.break`
: Break out of a `while` loop. Be sure not to confuse this with the`br` request (causing a line break).

`.continue`
: Finish the current iteration of a `while` loop, immediately restarting the next iteration.

5.21. Writing Macros

A *macro* is a collection of text and embedded commands that can be invoked multiple times. Use macros to define common operations. See Strings, for a (limited) alternative syntax to call macros.

Although the following requests can be used to create macros, simply using an undefined macro will cause it to be defined as empty. See Identifiers.

`.de` *name* [*end*]
`.de1` *name* [*end*]
`.dei` *name* [*end*]
`.dei1` *name* [*end*]
: Define a new macro named*name*. `gtroff` copies subsequent lines (starting with the next one) into an internal buffer until it encounters the line '. .' (two dots). If the optional second argument to `de` is present it is used as the macro closure request instead of '. .'.

  There can be spaces or tabs after the first dot in the line containing the ending token (either '.' or macro '*end*'). Don't insert a tab character immediately after the '. .',

otherwise it isn't recognized as the end-of-macro symbol.[54]

Here is a small example macro called 'P' that causes a break and inserts some vertical space. It could be used to separate paragraphs.

```
.de P
.  br
.  sp .8v
..
```

The following example defines a macro within another. Remember that expansion must be protected twice; once for reading the macro and once for executing.

```
\# a dummy macro to avoid a warning
.de end
..
.
.de foo
.  de bar end
.    nop \f[B]Hello \\\\$1!\f[]
.  end
..
.
.foo
.bar Joe
    ⇒ Hello Joe!
```

Since `\f` has no expansion, it isn't necessary to protect its backslash. Had we defined another macro within `bar` that takes a parameter, eight backslashes would be necessary before '`$1`'.

The `de1` request turns off compatibility mode while executing the macro. On entry, the current compatibility mode is saved and restored at exit.

```
.nr xxx 12345
.
.de aa
The value of xxx is \\n[xxx].
..
.de1 bb
The value of xxx is \\n[xxx].
..
.
.cp 1
.
```

[54] While it is possible to define and call a macro '.' with

```
.de .
.  tm foo
..
.
..    \" This calls macro '.'!
```

you can't use this as the end-of-macro macro: during a macro definition, '. .' is never handled as a call to '.', even if you say '`.de foo .`' explicitly.

```
.aa
    ⇒ warning: number register '[' not defined
    ⇒ The value of xxx is 0xxx].
.bb
    ⇒ The value of xxx is 12345.
```

The `dei` request defines a macro indirectly. That is, it expands strings whose names are *name* or *end* before performing the append.

This:

```
.ds xx aa
.ds yy bb
.dei xx yy
```

is equivalent to:

```
.de aa bb
```

The `dei1` request is similar to `dei` but with compatibility mode switched off during execution of the defined macro.

If compatibility mode is on, `de` (and `dei`) behave similar to `de1` (and `dei1`): A 'compatibility save' token is inserted at the beginning, and a 'compatibility restore' token at the end, with compatibility mode switched on during execution. See gtroff Internals, for more information on switching compatibility mode on and off in a single document.

Using `trace.tmac`, you can trace calls to `de` and `de1`.

Macro identifiers share their name space with identifiers for strings, diversions, and boxes; Identifiers.

See the description of the `als` request, for possible pitfalls if redefining a macro that has been aliased.

`.am` *name* [*end*]
`.am1` *name* [*end*]
`.ami` *name* [*end*]
`.ami1` *name* [*end*]

Works similarly to `de` except it appends onto the macro named*name*. So , to make the previously defined '`P`' macro set indented instead of block paragraphs, add the necessary code to the existing macro.

```
.am P
.ti +5n
..
```

The `am1` request turns off compatibility mode while executing the appended macro piece. To be more precise, a*compatibility sa ve* input token is inserted at the beginning of the appended code, and a*compatibility restore* input tok en at the end.

The `ami` request appends indirectly, meaning that `gtroff` expands strings whose names are*name* or *end* bef ore performing the append.

The `ami1` request is similar to `ami` but compatibility mode is switched off during execution of the defined macro.

Using `trace.tmac`, you can trace calls to `am` and `am1`.

See Strings, for the `als` and `rn` request to create an alias and rename a macro, respectively.

The `am`, `as`, `da`, `de`, `di`, and `ds` requests (together with their variants) only create a new object if the name of the macro, diversion, or string is currently undefined or if it is defined as a request; normally, they modify the value of an existing object.

`.return` [*anything*]
: Exit a macro, immediately returning to the caller.

  If called with an argument, exit twice, namely the current macro and the macro one level higher. This is used to define a wrapper macro for `return` in `trace.tmac`.

5.21.1. Copy Mode

When GNU `troff` processes certain requests, most importantly those which define a macro, string, or diversion, it does so in*cop y mode*: it copies the characters of the definition into a dedicated storage region, interpolating the escape sequences `\n`, `\$`, and `\*`, intepreting `\\` and `\RET` immediately and storing all other escape sequences in an encoded form.

Since the escape character escapes itself, you can control whether any escape sequence is interpreted at definition time or when it is later invoked or interpolated by selectively insulating the escapes with an extra backslash.[55]

```
.nr x 20
.de y
.nr x 10
\&\nx
\&\\nx
..
.y
    ⇒ 20 10
```

The counterpart to copy mode—a `roff` program's behavior when not defining a macro, string or diversion—where escapes are interpolated, requests invoked, and macros called immediately upon recognition, can be termed*interpretation mode*.

5.21.2. Parameters

The arguments to a macro or string can be examined using a variety of escapes.

`\n[.$]`
: The number of arguments passed to a macro or string. This is a read-only register.

  The `shift` request can change its value.

Any individual argument can be retrieved with one of the following escapes:

`\$`*n*
`\$(`*nn*
`\$[`*nnn*`]`
: Retrieve the*n* th, *nn* th or*nnn* th argument. As usual, the first f orm only accepts a single number (larger than zero), the second a two-digit number (larger than or equal

[55] Compare this to the `\def` and `\edef` commands in TEX.

to 10), and the third any positive integer value (larger than zero). Macros and strings can have an unlimited number of arguments. Because string and macro definitions are read in copy mode, use two backslashes on these in practice to prevent their interpolation until the macro is actually invoked.

`.shift` [*n*]
Shift the arguments 1 position, or as many positions as specified by its argument. After executing this request, argument*i* becomes argument *i*- *n*; arguments 1 to*n* are no longer available. Shifting by negative amounts is currently undefined.

The register `.$` is adjusted accordingly.

`\$*`
`\$@`
In some cases it is convenient to use all of the arguments at once (for example, to pass the arguments along to another macro). The `\$*` escape concatenates all the arguments separated by spaces. A similar escape is`\$@`, which concatenates all the arguments with each surrounded by double quotes, and separated by spaces. If not in compatibility mode, the input level of double quotes is preserved (see Request and Macro Arguments).

`\$^`
Handle the parameters of a macro as if they were an argument to the `ds` or similar requests.

```
.de foo
.  tm $1=‘\\$1’
.  tm $2=‘\\$2’
.  tm $*=‘\\$*’
.  tm $@=‘\\$@’
.  tm $^=‘\\$^’
..
.foo " This is a "test"
    ⇒ $1=‘ This is a ’
    ⇒ $2=‘test"’
    ⇒ $*=‘ This is a  test"’
    ⇒ $@=‘" This is a " "test""’
    ⇒ $^=‘" This is a "test"’
```

This escape is useful mainly for macro packages like `trace.tmac`, which redefines some requests and macros for debugging purposes.

`\$0`
The name used to invoke the current macro. The`als` request can mak e a macro have more than one name.

If a macro is called as a string (within another macro), the value of `\$0` isn’t changed.

```
.de foo
.  tm \\$0
..
.als 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
```

See Request and Macro Arguments.

5.22. Page Motions

See Manipulating Spacing, for a discussion of the main request for vertical motion, `sp`.

`.mk` [*reg*]
`.rt` [*dist*]

The request `mk` can be used to mark a location on a page, for movement to later. This request takes a register name as an argument in which to store the current page location. With no argument it stores the location in an internal register. The results of this can be used later by the `rt` or the `sp` request (or the `\v` escape).

The `rt` request returns *upwards* to the location marked with the last `mk` request. If used with an argument, return to a position which distance from the top of the page is *dist* (no previous call to `mk` is necessary in this case). Default scaling indicator is 'v'.

If a page break occurs between a `mk` request and its matching `rt` request, the `rt` is silently ignored.

Here a primitive solution for a two-column macro.

```
.nr column-length 1.5i
.nr column-gap 4m
.nr bottom-margin 1m
.

.de 2c
.  br
.  mk
```

```
.  ll \\n[column-length]u
.  wh -\\n[bottom-margin]u 2c-trap
.  nr right-side 0
..
.

.de 2c-trap
.  ie \\n[right-side] \{\
.    nr right-side 0
.    po -(\\n[column-length]u + \\n[column-gap]u)
.    \" remove trap
.    wh -\\n[bottom-margin]u
.  \}
.  el \{\
.    \" switch to right side
.    nr right-side 1
.    po +(\\n[column-length]u + \\n[column-gap]u)
.    rt
.  \}
..
.

.pl 1.5i
.ll 4i
This is a small test that shows how the
rt request works in combination with mk.

.2c
Starting here, text is typeset in two columns.
Note that this implementation isn't robust
and thus not suited for a real two-column
macro.
```

Result:

```
This is a small test that shows how the
rt request works in combination with mk.

Starting  here,    isn't    robust
text is typeset    and   thus  not
in two columns.    suited  for   a
Note that  this    real two-column
implementation     macro.
```

The following escapes give fine control of movements about the page.

`\v'`*e*`'`

Move vertically, usually from the current location on the page (if no absolute position operator '`|`' is used). The argument*e* specifies the distance to mo ve; positive is downwards and negative upwards. The default scaling indicator for this escape is '`v`'. Beware, however, that `gtroff` continues text processing at the point where the motion ends, so you should always balance motions to avoid interference with text

processing.

`\v` doesn't trigger a trap. This can be quite useful; for example, consider a page bottom trap macro that prints a marker in the margin to indicate continuation of a footnote or something similar.

There are some special-case escapes for vertical motion.

`\r`
: Move upwards 1 v.

`\u`
: Move upwards .5 v.

`\d`
: Move down .5 v.

`\h'`*e*`'`
: Move horizontally, usually from the current location (if no absolute position operator '`|`' is used). The expression *e* indicates how far to move: positive is rightwards and negative leftwards. The default scaling indicator for this escape is '`m`'.

  This horizontal space is not discarded at the end of a line. To insert discardable space of a certain length use the `ss` request.

There are a number of special-case escapes for horizontal motion.

`\SP`
: An unbreakable and unpaddable (i.e. not expanded during filling) space. (Note: This is a backslash followed by a space.)

`\~`
: An unbreakable space that stretches like a normal inter-word space when a line is adjusted.

`\|`
: A 1/6 th em unbreakable space. Ignored for TTY output devices (rounded to zero).

  However, if there is a glyph defined in the current font file with name `\|` (note the leading backslash), the width of this glyph is used instead (even for TTYs).

`\^`
: A 1/12 th em unbreakable space. Ignored for TTY output devices (rounded to zero).

  However, if there is a glyph defined in the current font file with name `\^` (note the leading backslash), the width of this glyph is used instead (even for TTYs).

`\0`
: An unbreakable space the size of a digit.

The following string sets the TEX logo:

```
.ds TeX T\h'-.1667m'\v'.224m'E\v'-.224m'\h'-.125m'X
```

`\w'`*text*`'`
`\n[st]`
`\n[sb]`
`\n[rst]`
`\n[rsb]`
`\n[ct]`
`\n[ssc]`

`\n[skw]`
: Return the width of the specified *text* in basic units. This allows horizontal movement based on the width of some arbitrary text (e.g. given as an argument to a macro).

```
The length of the string ‘abc’ is \w’abc’u.
    ⇒ The length of the string ‘abc’ is 72u.
```

Font changes may occur in *text*, which don't affect current settings.

After use, `\w` sets several registers:

`st`
`sb`
: The highest and lowest point of the baseline, respectively, in *text*.

`rst`
`rsb`
: Like the `st` and `sb` registers, but takes account of the heights and depths of glyphs. In other words, this gives the highest and lowest point of *text*. Values below the baseline are negative.

`ct`
: Defines the kinds of glyphs occurring in *text*:

  0 — only short glyphs, no descenders or tall glyphs.

  1 — at least one descender.

  2 — at least one tall glyph.

  3 — at least one each of a descender and a tall glyph.

`ssc`
: The amount of horizontal space (possibly negative) that should be added to the last glyph before a subscript.

`skw`
: How far to right of the center of the last glyph in the `\w` argument, the center of an accent from a roman font should be placed over that glyph.

`\k`*p*
`\k(`*ps*
`\k[`*position*`]`
: 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]`
: The current horizontal position at the input line.

`\n[.k]`
: A read-only register containing the current horizontal output position (relative to the current indentation).

`\o'`*abc*`'`
: Overstrike glyphs *a*, *b*, *c*, ...; the glyphs are centered, and the resulting spacing is the largest width of the affected glyphs.

`\z`*g*
: Print glyph *g* with zero width, i.e., without spacing. Use this to overstrike glyphs left-aligned.

`\Z'`*anything*`'`
: Print *anything*, then restore the horizontal and vertical position. The argument may not contain tabs or leaders.

  The following is an example of a strike-through macro:

```
.de ST
.nr ww \w'\\$1'
\Z@\v'-.25m'\l'\\n[ww]u'@\\$1
..
.
This is
.ST "a test"
an actual emergency!
```

5.23. Drawing Requests

`gtroff` provides a number of ways to draw lines and other figures on the page. Used in combination with the page motion commands (see Page Motions), a wide variety of figures can be drawn. However, for complex drawings these operations can be quite cumbersome, and it may be wise to use graphic preprocessors like `gpic` or `ggrn`. See `gpic`, and `ggrn`.

All drawing is done via escapes.

`\l'`*l*`'`
`\l'`*lg*`'`
: Draw a line horizontally. *l* is the length of the line to be drawn. If it is positive, start the line at the current location and draw to the right; its end point is the new current location. Negative values are handled differently: The line starts at the current location and draws to the left, but the current location doesn't move.

  l can also be specified absolutely (i.e. with a leading '`|`'), which draws back to the beginning of the input line. Default scaling indicator is '`m`'.

  The optional second parameter *g* is a glyph to draw the line with. If this second argument is not specified, `gtroff` uses the underscore glyph, `\[ru]`.

  To separate the two arguments (to prevent `gtroff` from interpreting a drawing glyph as a scaling indicator if the glyph is represented by a single character) use `\&`.

```
.de box
\[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 escapes both draw from the current location to the beginning of the*input* line—this w orks because the line length is negative, not moving the current point.

`\L'`*l*`'`
`\L'`*lg*`'`
: Draw vertical lines. Its parameters are similar to the `\l` escape, except that the default scaling indicator is '`v`'. The movement is downwards for positive values, and upwards for negative values. The default glyph is the box rule glyph, `\[br]`. As with the vertical motion escapes, text processing blindly continues where the line ends.

```
This is a \L'3v'test.
```

Here is the result, produced with `grotty`.

```
This is a
          |
          |
          |test.
```

`\D'`*command arg ...*`'`

The `\D` escape provides a variety of drawing functions. On character devices, only vertical and horizontal lines are supported within `grotty`; other devices may only support a subset of the available drawing functions.

The default scaling indicator for all subcommands of `\D` is '`m`' for horizontal distances and '`v`' for vertical ones. Exceptions are '`\D'f ...'`' and '`\D't ...'`', which use `u` as the default, and '`\D'Fx ...'`', which arguments are treated similar to the `defcolor` request.

`\D'l` *dx dy*`'`

Draw a line from the current location to the relative point specified by (*dx*,*dy*), where positive values mean right and down, respectively. The end point of the line is the new current location.

The following example is a macro for creating a box around a text string; for simplicity, the box margin is taken as a fixed value, 0.2 m.

```
.de BOX
.  nr @wd \w'\\$1'
\h'.2m'\
\h'-.2m'\v'(.2m - \\n[rsb]u)'\
\D'l 0 -(\\n[rst]u - \\n[rsb]u + .4m)'\
\D'l (\\n[@wd]u + .4m) 0'\
\D'l 0 (\\n[rst]u - \\n[rsb]u + .4m)'\
\D'l -(\\n[@wd]u + .4m) 0'\
\h'.2m'\v'-(.2m - \\n[rsb]u)'\
\\$1\
\h'.2m'
..
```

First, the width of the string is stored in register `@wd`. Then, four lines are drawn to form a box, properly offset by the box margin. The registers `rst` and `rsb` are set by the `\w` escape, containing the largest height and depth of the whole string.

`\D'c` *d*`'` Draw a circle with a diameter of*d* with the leftmost point at the current position. After drawing, the current location is positioned at the rightmost point of the circle.

`\D'C` *d*`'` Draw a solid circle with the same parameters and behaviour as an outlined circle. No outline is drawn.

`\D'e` *x y*`'`

Draw an ellipse with a horizontal diameter of *x* and a vertical diameter of *y* with the leftmost point at the current position. After drawing,

the current location is positioned at the rightmost point of the ellipse.

`\D'E` *x y*`'`
: Draw a solid ellipse with the same parameters and behaviour as an outlined ellipse. No outline is drawn.

`\D'a` *dx1 dy1 dx2 dy2*`'`
: Draw an arc clockwise from the current location through the two specified relative locations (*dx1*,*dy1*) and (*dx2*,*dy2*). The coordinates of the first point are relative to the current position, and the coordinates of the second point are relative to the first point. After drawing, the current position is moved to the final point of the arc.

`\D'~` *dx1 dy1 dx2 dy2 ...*`'`
: Draw a spline from the current location to the relative point (*dx1*,*dy1*) and then to (*dx2*,*dy2*), and so on. The current position is moved to the terminal point of the drawn curve.

`\D'f` *n*`'` Set the shade of gray to be used for filling solid objects to*n*; *n* m ust be an integer between 0 and 1000, where 0 corresponds solid white and 1000 to solid black, and values in between correspond to intermediate shades of gray. This applies only to solid circles, solid ellipses, and solid polygons. By default, a level of 1000 is used.

Nonintuitively, the current point is moved horizontally to the right by *n*.

Don't use this command! It has the serious drawback that it is always rounded to the next integer multiple of the horizontal resolution (the value of the `hor` keyword in the `DESC` file). Use `\M` (see Colors) or '`\D'Fg ...`'' instead.

`\D'p` *dx1 dy1 dx2 dy2 ...*`'`
: Draw a polygon from the current location to the relative position (*dx1*,*dy1*) and then to (*dx2*,*dy2*) and so on. When the specified data points are exhausted, a line is drawn back to the starting point. The current position is changed by adding the sum of all arguments with odd index to the actual horizontal position and the even ones to the vertical position.

`\D'P` *dx1 dy1 dx2 dy2 ...*`'`
: Draw a solid polygon with the same parameters and behaviour as an outlined polygon. No outline is drawn.

Here a better variant of the box macro to fill the box with some color. The box must be drawn before the text since colors in GNU `troff` are not transparent; the filled polygon would hide the text completely.

```
.de BOX
.  nr @wd \w'\\$1'
\h'.2m'\
\h'-.2m'\v'(.2m - \\n[rsb]u)'\
\M[lightcyan]\
\D'P 0 -(\\n[rst]u - \\n[rsb]u + .4m) \
```

```
        (\\n[@wd]u + .4m) 0 \
        0 (\\n[rst]u - \\n[rsb]u + .4m) \
        -(\\n[@wd]u + .4m) 0'\
\h'.2m'\v'-(.2m - \\n[rsb]u)'\
\M[]\
\\$1\
\h'.2m'
..
```

If you want a filled polygon that has exactly the same size as an unfilled one, you must draw both an unfilled and a filled polygon. A filled polygon is always smaller than an unfilled one because the latter uses straight lines with a given line thickness to connect the polygon's corners, while the former simply fills the area defined by the coordinates.

```
\h'1i'\v'1i'\
\# increase line thickness
\Z'\D't 5p"\
\# draw unfilled polygon
\Z'\D'p 3 3 -6 0"\
\# draw filled polygon
\Z'\D'P 3 3 -6 0"
```

`\D't n'` Set the current line thickness to *n* basic units . A value of zero selects the smallest available line thickness. A negative value makes the line thickness proportional to the current point size (this is the default behaviour of AT&T `troff`).

Nonintuitively, the current point is moved horizontally to the right by *n*.

`\D'F`*scheme color_components*`'`

Change current fill color. *scheme* is a single letter denoting the color scheme: '`r`' (rgb), '`c`' (cmy), '`k`' (cmyk), '`g`' (gray), or '`d`' (default color). The color components use exactly the same syntax as in the `defcolor` request (see Colors); the command `\D'Fd'` doesn't take an argument.

No position changing!

Examples:

```
\D'Fg .3'      \" same gray as \D'f 700'
\D'Fr #0000ff' \" blue
```

See Graphics Commands.

`\b'`*string*`'`

Pile a sequence of glyphs vertically, and center it vertically on the current line. Use it to build large brackets and braces.

Here an example how to create a large opening brace:

```
\b'\[lt]\[bv]\[lk]\[bv]\[lb]'
```

The first glyph is on the top, the last glyph in *string* is at the bottom. GNU `troff` separates the glyphs vertically by 1 m, and the whole object is centered 0.5 m above the current baseline; the largest glyph width is used as the width for the whole object. This rather inflexible positioning algorithm doesn't work with `-Tdvi` since the bracket pieces vary in height for this device. Instead, use the `eqn` preprocessor.

See Manipulating Spacing, how to adjust the vertical spacing with the `\x` escape.

5.24. Traps

Traps are locations in the output, or conditions on the input that, when reached or fulfilled, cause a specified macro to be called. 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 certain number of input lines, or at the end of input. Macros invoked by traps have no arguments. Setting a trap is also called *planting*. It is also said that a trap is *sprung* if the associated macro is executed.

5.24.1. 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. They have a variety of purposes.

- setting headers and footers
- setting body text in multiple columns
- setting footnotes

The location parameter used in vertical position traps has a default scaling indicator of 'v', and its value is rounded to be multiples of the vertical resolution (as given in register `.V`).

`.vpt` [*flag*]
`\n[.vpt]`
Enable vertical position traps if *flag* is non-z ero 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 setting of this is available in the `.vpt` read-only register.

A page can't be ejected if `vpt` is set to zero.

5.24.1.1. Page Location Traps

`.wh` *dist* [*name*]
Call macro *name* when the v ertical position *dist* on the page is reached or passed in the downward direction. 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 remo ved; this is `wh`'s sole function if *name* is missing.

A trap is sprung only if it is *visib le*, meaning that its location is reachable on the page[56] and it is not hidden by another trap at the same location already planted

[56] A trap planted at '`20i`' or '`-30i`' will not be sprung on a page of length '`11i`'.

there.

An example of how a macro package might set headers and footers follows.

```
.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; the y 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 Debugging). 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]`
: 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*]
: Change the location of a trap by moving macro*name* to ne w location*dist*, or b y unplanting it altogether if*dist* is absent. P arameters to `ch` are specified in the opposite order from `wh`. If*name* is the earliest planted macro of m ultiple traps at the same location, (re)moving it from that location exposes the macro next least recently planted at the same place.[57]

  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` device) illustrates this behavior.[58] 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
```

[57] 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.

[58] ...which is compatible with Heirloom Doctools `troff`.

```
.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]`

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 Page Control.

Since the `.ne` register is set only by traps it doesn't make much sense to use it outside of trap macros.

`\n[.trunc]`

A read-only register containing the amount of vertical space truncated from an `sp` request by the most recently sprung vertical position trap, or, if the trap was sprung by an `ne` request, minus the amount of vertical motion produced by the `ne` request. In other words, at the point a trap is sprung, it represents the difference of what the vertical position would have been but for the trap, and what the vertical position actually is.

Since the `.trunc` register is only set by traps it doesn't make much sense to use it outside of trap macros.

`\n[.pe]`

A read-only register that is set to 1 while a page is ejected with the `bp` request (or by the end of input).

Outside of traps this register is always zero. In the following example, only the second call to `x` is caused by `bp`.

```
.de x
\&.pe=\\n[.pe]
.br
..
.wh 1v x
.wh 4v x
A line.
.br
Another line.
.br
```

```
⇒ A line.
  .pe=0
  Another line.

  .pe=1
```

An important fact to consider while designing macros is that diversions and traps do not interact normally. For example, if a trap invokes a header macro (while outputting a diversion) that tries to change the font on the current page, the effect is not visible before the diversion has completely been printed (except for input protected with `\!` or `\?`) since the data in the diversion is already formatted. In most cases, this is not the expected behaviour.

5.24.1.2. Diversion Traps

`.dt` [*dist name*]
: Set a trap *within* a diversion at location*dist*. The location is interpreted relativ e to diversion rather than page boundaries. If called with fewer than two arguments, the diversion trap is removed.

  There exists only a single diversion trap.

  The register `.t` works within diversions. See Diversions.

5.24.2. Input Line Traps

`.it` *n name*
`.itc` *n name*
: Set an input line trap, calling macro*name* after the ne xt *n* lines of text input have been read. Lines beginning with the control character or no-break control character are not counted.

  Consider a macro '`.B` *n*' which sets the next *n* input lines in bold.

```
.de B
.  it \\$1 EB
.  ft B
..
.de EB \" end bold
.  ft R
..
```

  With `itc`, interrupted text lines are not counted separately.

```
.de Monospace
.  it \\$1 End-Monospace
.  fam C
..
.de End-Monospace
.  fam T
..
Syntax:
.Monospace 1
```

```
.ft B
mycommand \c
[\c
.ft I
operand \c
]
```

Both requests are associated with the current environment (see Environments); switching to another environment disables the current input trap, and going back reactivates it, restoring the count of already processed lines.

5.24.3. Blank Line Traps

`.blm` [*name*]
: Set a blank line trap, calling the macro*name* when GNU `troff` encounters a b lank line in an input file, instead of the usual behavior (see Breaking). 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.

5.24.4. Leading Space Traps

`.lsm` [*name*]
`\n[lsn]`
`\n[lss]`
: 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 Breaking).

  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*.

5.24.5. End-of-input Traps

`.em` [*name*]
: 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 (and for compatibility with AT&T `troff`), the end-of-input macro exits as soon as it causes a page break and no remaining data is in the partially collected line.

Let us assume that there is no `\c` in the above `approval` macro, and that the page is full and has been ended with, say, a `br` request. The `ne` request now causes the start of a new page, which in turn makes `troff` exit immediately for the reasons just described. In most situations this is not intended.

To force processing of the whole end-of-input macro independently of this behavior, it is thus advisable to insert something that starts an empty partially collected line (`\c`) whenever there is a chance that a page break can happen. In the above example, the call of the `ne` request assures that the remaining code stays on the same page, so we have to insert `\c` only once.

The next example shows how to append three lines, then 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.

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,[59] 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.

5.25. Diversions

In `roff` systems it is possible to format text as if for output, but instead of writing it immediately, one can*div ert* the formatted text into a named storage area. The same name space is used for such*div ersions* as for strings and macros; Identifiers. 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*unf ormatted* input text, and the latter capture *formatted* output. 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 of type '`mac`' to be emitted (see Debugging). A diversion does not exist for the purpose of testing with the `d` conditional operator until it ends (see Operators in Conditionals). The `.z` register can be used to test the identity of the current diversion. The following requests are used to create and alter diversions.

`.di` *macro*
`.da` *macro*

Begin a diversion. Like the `de` request, it takes an argument of a macro name to divert subsequent text into. The`da` macro appends to an e xisting diversion.

`di` or `da` without an argument ends the diversion.

The current partially filled line is included into the diversion. See the`box` request below for an example. Switching to another (empty) environment (with the `ev` request)

[59] Another, taken by the `groff man` macros, is to intercept `ne` requests and wrap `bp` ones.

avoids the inclusion of the current partially filled line; Environments.

`.box` *macro*
`.boxa` *macro*

Begin (or append to) a diversion like the `di` and `da` requests. The difference is that `box` and `boxa` do not include a partially filled line in the diversion.

Compare this:

```
Before the box.
.box xxx
In the box.
.br
.box
After the box.
.br
    ⇒ Before the box.  After the box.
.xxx
    ⇒ In the box.
```

with this:

```
Before the diversion.
.di yyy
In the diversion.
.br
.di
After the diversion.
.br
    ⇒ After the diversion.
.yyy
    ⇒ Before the diversion.  In the diversion.
```

`box` or `boxa` without an argument ends the diversion.

`\n[.z]`
`\n[.d]`

Diversions may be nested. The read-only register `.z` contains the name of the current diversion (this is a string-valued register). The read-only register `.d` contains the current vertical place in the diversion. If not in a diversion, it is the same as register `nl`.

`\n[.h]`

The 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.[60]

```
.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]
```

[60] Thus, the “water” gets “higher” proceeding*do wn* the page.

```
⇒ .h==40, nl==120
```

As the previous example shows, empty lines are not considered in the return value of the `.h` register.

`\n[dn]`
`\n[dl]`

After completing a diversion, the writable registers `dn` and `dl` contain the vertical and horizontal size of the diversion. Only the just-processed lines are counted: for the computation of `dn` and `dl`, the requests `da` and `boxa` are handled as if `di` and `box` had been used—lines that have been already stored in a macro are not taken into account.

```
.\" Center text both horizontally and vertically.
.
.\" Disable the escape character with .eo so that we
.\" don't have to double backslashes on the \n escapes.
.eo
.\" Macro .(c starts centering mode.
.de (c
.  br
.  ev (c
.  evc 0
.  in 0
.  nf
.  di @c
..

.\" Macro .)c terminates centering mode.
.de )c
.  br
.  ev
.  di
.  nr @s (((\n[.t]u - \n[dn]u) / 2u) - 1v)
.  sp \n[@s]u
.  ce 1000
.  @c
.  ce 0
.  sp \n[@s]u
.  br
.  fi
.  rr @s
.  rm @c
..
.\" End of macro definitions; restore escape character.
.ec
```

`\!`
`\?`*anything*`\?`

Prevent requests, macros, and escapes from being interpreted when read into a diversion. Both escapes take the given text and*tr ansparently* embed it into the diversion. This is useful for macros that shouldn't be invoked until the diverted text is

actually output.

The \! escape transparently embeds text up to and including the end of the line. The \? escape transparently embeds text until the next occurrence of the \? escape. Example:

```
\?anything\?
```

anything may not contain newlines; use \! to embed newlines in a diversion. The escape sequence \? is also recognized in copy mode and turned into a single internal code; it is this code that terminates*an ything*. Thus the following example prints 4.

```
.nr x 1
.nf
.di d
\?\\?\\\\?\\\\\\\\nx\\\\?\\?\?
.di
.nr x 2
.di e
.d
.di
.nr x 3
.di f
.e
.di
.nr x 4
.f
```

Both escapes read the data in copy mode.

If \! is used in the top-level diversion, its argument is directly embedded into the gtroff intermediate output. This can be used for example to control a postprocessor that processes the data before it is sent to the device driver.

The \? escape used in the top-level diversion produces no output at all; its argument is simply ignored.

`.output` *string*

Emit *string* directly to the gtroff intermediate output (subject to copy mode interpretation); this is similar to \! used at the top level. An initial double quote in*string* is stripped off to allow initial blanks.

This request can't be used before the first page has started—if you get an error, simply insert .br before the output request.

Without argument, output is ignored.

Use with caution! It is normally only needed for mark-up used by a postprocessor that does something with the output before sending it to the output device, filtering out *string* again.

`.asciify` *div*

Unformat the diversion *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 lik e ordinary

input characters when*div* is reread. Doing so can be useful in conjunction with the `writem` request. `asciify` can be also used for gross hacks; for example, the following sets register `n` to 1.

```
.tr @.
.di x
@nr n 1
.br
.di
.tr @@
.asciify x
.x
```

`asciify` cannot return all items in a diversion back to their source equivalent; nodes such as those produced by `\N[...]` will remain nodes, so the result cannot be guaranteed to be a pure string.

See Copy Mode.

`.unformat` *div*

Like `asciify`, unformat the diversion *div*. However, `unformat` handles only tabs and spaces between words, the latter usually arising from spaces or newlines in the input. Tabs are treated as input tokens, and spaces become stretchable again.

The vertical sizes of lines are not preserved, but glyph information (font, font size, space width, etc.) is retained. `unformat` can be useful in conjunction with the `box` and `boxa` requests.

5.26. Environments

It happens frequently that some text should be printed in a certain format regardless of what may be in effect at the time, for example, in a trap invoked macro to print headers and footers. To solve this `gtroff` processes text in*en vironments*. An environment contains most of the parameters that control text processing. It is possible to switch amongst these environments; by default `gtroff` processes text in environment 0. The following is the information kept in an environment.

- font parameters (size, family, style, glyph height and slant, space and inter-sentence space size)
- page parameters (line length, title length, vertical spacing, line spacing, indentation, line numbering, centering, right-justifying, underlining, hyphenation data)
- fill and adjust mode
- tab stops, tab and leader characters, escape character, no-break and hyphen indicators, margin character data
- partially collected lines
- input traps
- drawing and fill colours

These environments may be given arbitrary names (see Identifiers.) Old versions of `troff` only had environments named ‘`0`’, ‘`1`’, and ‘`2`’.

`.ev` [*env*]
`\n[.ev]`

Switch to another environment. The argument *env* is the name of the environment to switch to. With no argument, `gtroff` switches back to the previous environment. There is no limit on the number of named environments; they are created the first time that they are referenced. The `.ev` read-only register contains the name or number of the current environment. This is a string-valued register.

A call to `ev` (with argument) pushes the previously active environment onto a stack. If, say, environments '`foo`', '`bar`', and '`zap`' are called (in that order), the first `ev` request without parameter switches back to environment '`bar`' (which is popped off the stack), and a second call switches back to environment '`foo`'.

Here is an example:

```
.ev footnote-env
.fam N
.ps 6
.vs 8
.ll -.5i
.ev

...

.ev footnote-env
\(dg Note the large, friendly letters.
.ev
```

`.evc` *env*

Copy the environment *env* into the current environment.

The following environment data is not copied:

- Partially filled lines.
- The status whether the previous line was interrupted.
- The number of lines still to center, or to right-justify, or to underline (with or without underlined spaces); they are set to zero.
- The status whether a temporary indentation is active.
- Input traps and its associated data.
- Line numbering mode is disabled; it can be reactivated with '`.nm +0`'.
- The number of consecutive hyphenated lines (set to zero).

`\n[.w]`
`\n[.cht]`
`\n[.cdp]`
`\n[.csk]`

The `\n[.w]` register contains the width of the last glyph added to the current environment.

The `\n[.cht]` register contains the height of the last glyph added to the current environment.

The `\n[.cdp]` register contains the depth of the last glyph added to the current

environment. It is positive for glyphs extending below the baseline.

The `\n[.csk]` register contains the *skew* (how far to the right of the glyph's center that `gtroff` should place an accent) of the last glyph added to the current environment.

`\n[.n]`
The `\n[.n]` register contains the length of the previous output line in the current environment.

5.27. Suppressing output

`\O`*num*
Disable or enable output depending on the value of *num*:

'`\O0`' Disable any glyphs from being emitted to the device driver, provided that the escape occurs at the outer level (see `\O[3]` and `\O[4]`). Motion is not suppressed so effectively `\O[0]` means *pen up*.

'`\O1`' Enable output of glyphs, provided that the escape occurs at the outer level.

`\O0` and `\O1` also reset the four registers '`opminx`', '`opminy`', '`opmaxx`', and '`opmaxy`' to −1. See Register Index. These four registers mark the top left and bottom right hand corners of a box that encompasses all written glyphs.

For example the input text:

```
Hello \O[0]world \O[1]this is a test.
```

produces the following output:

```
Hello       this is a test.
```

'`\O2`' Provided that the escape occurs at the outer level, enable output of glyphs and also write out to `stderr` the page number and four registers encompassing the glyphs previously written since the last call to `\O`.

'`\O3`' Begin a nesting level. At start-up, `gtroff` is at outer level. The current level is contained within the read-only register `.O`. See Built-in Registers.

'`\O4`' End a nesting level. The current level is contained within the read-only register `.O`. See Built-in Registers.

'`\O[5`*P*`filename]`'
This escape is `grohtml` specific. Provided that this escape occurs at the outer nesting level write the `filename` to `stderr`. The position of the image, *P*, must be specified and must be one of `l`, `r`, `c`, or `i` (left, right, centered, inline). *filename* is associated with the production of the next inline image.

5.28. Colors

`.color` [*n*]
`\n[.color]`

If *n* is missing or non-zero, activate colors (this is the default); otherwise, turn it off.

The read-only register `.color` is 1 if colors are active, 0 otherwise.

Internally, `color` sets a global flag; it does not produce a token. Similar to the`cp` request, you should use it at the beginning of your document to control color output.

Colors can be also turned off with the `-c` command-line option.

`.defcolor` *ident scheme color_components*

Define color with name*ident*. *scheme* can be one of the f ollowing values: `rgb` (three components), `cmy` (three components), `cmyk` (four components), and `gray` or `grey` (one component).

Color components can be given either as a hexadecimal string or as positive decimal integers in the range 0–65535. A hexadecimal string contains all color components concatenated. It must start with either `#` or `##`; the former specifies hex values in the range 0–255 (which are internally multiplied by 257), the latter in the range 0–65535. Examples: `#FFC0CB` (pink), `##ffff0000ffff` (magenta). The default color name value is device-specific (usually black). It is possible that the default color for `\m` and `\M` is not identical.

A new scaling indicator `f` has been introduced, which multiplies its value by 65536; this makes it convenient to specify color components as fractions in the range 0 to 1 (1f equals 65536u). Example:

```
.defcolor darkgreen rgb 0.1f 0.5f 0.2f
```

Note that `f` is the default scaling indicator for the `defcolor` request, thus the above statement is equivalent to

```
.defcolor darkgreen rgb 0.1 0.5 0.2
```

`.gcolor` [*color*]
`\m`*c*
`\m(`*co*
`\m[`*color*`]`
`\n[.m]`

Set (glyph) drawing color. The following examples show how to turn the next four words red.

```
.gcolor red
these are in red
.gcolor
and these words are in black.
```

```
\m[red]these are in red\m[] and these words are in black.
```

The escape `\m[]` returns to the previous color, as does a call to `gcolor` without an argument.

The name of the current drawing color is available in the read-only, string-valued register '`.m`'.

The drawing color is associated with the current environment (see Environments).

`\m` doesn't produce an input token in GNU `troff`. As a consequence, it can be used in requests like `mc` (which expects a single character as an argument) to change the color on the fly:

```
.mc \m[red]x\m[]
```

`.fcolor` [*color*]
`\M`*c*
`\M(`*co*
`\M[`*color*`]`
`\n[.M]`

Set fill (background) color for filled objects drawn with the `\D'...'` commands.

A red ellipse can be created with the following code:

```
\M[red]\h'0.5i'\D'E 2i 1i'\M[]
```

The escape `\M[]` returns to the previous fill color, as does a call to `fcolor` without an argument.

The name of the current fill (background) color is available in the read-only, string-valued register '`.M`'.

The fill color is associated with the current environment (see Environments).

`\M` doesn't produce an input token in GNU `troff`.

5.29. I/O

`gtroff` has several requests for including files:

`.so` *file*

Read in the specified*file* and include it in place of the `so` request. This is quite useful for large documents, e.g. keeping each chapter in a separate file. See `gsoelim`, for more information.

Since `gtroff` replaces the `so` request with the contents of `file`, it makes a difference whether the data is terminated with a newline or not: Assuming that file `xxx` contains the word '`foo`' without a final newline, this

```
This is
.so xxx
bar
```

yields '`This is foobar`'.

The search path for *file* can be controlled with the `-I` command-line option.

`.pso` *command*

Read the standard output from the specified*command* and include it in place of the `pso` request.

This request causes an error if used in safer mode (which is the default). Use `groff`'s or `troff`'s `-U` option to activate unsafe mode.

The comment regarding a final newline for the `so` request is valid for `pso` also.

`.mso` *file*
: Identical to the `so` request except that `gtroff` searches for the specified *file* in the same directories as macro files for the `-m` command-line option. If the file name to be included has the form *name*`.tmac` and it isn't found, `mso` tries to include `tmac.`*name* and vice versa. If the file does not exist, a warning of type '`file`' is emitted. See Debugging, for information about warnings.

`.trf` *file*
`.cf` *file*
: Transparently output the contents of *file*. Each line is output as if it were preceded by `\!`; however, the lines are *not* subject to copy mode interpretation. If the file does not end with a newline, then a newline is added (`trf` only). For example, to define a macro `x` containing the contents of file `f`, use

```
.ev 1
.di x
.trf f
.di
.ev
```

: The calls to `ev` prevent that the current partial input line becomes part of the diversion.

: Both `trf` and `cf`, when used in a diversion, embeds an object in the diversion which, when reread, causes the contents of *file* to be transparently copied through to the output. In Unix `troff`, the contents of *file* is immediately copied through to the output regardless of whether there is a current diversion; this behaviour is so anomalous that it must be considered a bug.

: While `cf` copies the contents of *file* completely unprocessed, `trf` disallows characters such as NUL that are not valid `gtroff` input characters (see Identifiers).

: For `cf`, within a diversion, 'completely unprocessed' means that each line of a file to be inserted is handled as if it were preceded by `\!\\!`.

: Both requests cause a line break.

`.nx` [*file*]
: Force `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 ...*]]
: 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*

Pipe the output of `gtroff` to the shell command(s) specified by *pipe*. This request must occur before `gtroff` has a chance to print anything.

`pi` causes an error if used in safer mode (which is the default). Use `groff`'s or `troff`'s `-U` option to activate unsafe mode.

Multiple calls to `pi` are allowed, acting as a chain. For example,

```
.pi foo
.pi bar
...
```

is the same as '`.pi foo | bar`'.

The intermediate output format of GNU `troff` is piped to the specified commands. Consequently, calling `groff` without the `-Z` option normally causes a fatal error.

`.sy` *cmds*

`\n[systat]`

Execute the shell command(s) specified by *cmds*. The output is not saved anywhere, so it is up to the user to do so.

This request causes an error if used in safer mode (which is the default). Use

groff's or troff's -U option to activate unsafe mode.

For example, the following code fragment introduces the current time into a document:

```
.sy perl -e 'printf ".nr H %d\\n.nr M %d\\n.nr S %d\\n",\
             (localtime(time))[2,1,0]' > /tmp/x\n[$$]
.so /tmp/x\n[$$]
.sy rm /tmp/x\n[$$]
\nH:\nM:\nS
```

This works by having the Perl script (run by sy) print out the nr requests that set the registers H, M, and S, and then reading those commands in with the so request.

For most practical purposes, the registers seconds, minutes, and hours, which are initialized at start-up of GNU troff, should be sufficient. Use the af request to format their values for output.

```
.af hours 00
.af minutes 00
.af seconds 00
\n[hours]:\n[minutes]:\n[seconds]
```

The writable register systat contains the return value of the system() function executed by the last sy request.

.open *stream file*

.opena *stream file*

Open the specified*file* f or writing and associates the specified*stream* with it.

The opena request is like open, but if the file exists, append to it instead of truncating it.

Both open and opena cause an error if used in safer mode (which is the default). Use groff's or troff's -U option to activate unsafe mode.

.write *stream data*

.writec *stream data*

Write to the file associated with the specified*stream*. The stream m ust previously have been the subject of an open request. The remainder of the line is interpreted as the ds request reads its second argument: A leading '"' is stripped, and it is read in copy mode.

The writec request is like write, but only write appends a newline to the data.

.writem *stream xx*

Write the contents of the macro or string *xx* to the file associated with the specified *stream*.

xx is read in copy mode, i.e., already formatted elements are ignored. Consequently, diversions must be unformatted with the asciify request before calling writem. Usually, this means a loss of information.

.close *stream*

Close the specified*stream*; the stream is no longer an acceptab le 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*
\V(*ev*
\V[*env*]
: Interpolate the contents of the specified environment variable *env* (one-character name *e*, two-character name *ev*) as returned by the function `getenv`. `\V` is interpreted in copy mode.

5.30. Postprocessor Access

There are two escapes that give information directly to the postprocessor. This is particularly useful for embedding PostScript into the final document.

`.device` *xxx*
\X'*xxx*'
: Embeds its argument into the `gtroff` output preceded with '`x X`'.

  The escapes `\&`, `\)`, `\%`, and `\:` are ignored within `\X`, '`\ `' and `\~` are converted to single space characters. All other escapes (except `\\`, which produces a backslash) cause an error.

  Contrary to `\X`, the `device` request simply processes its argument in copy mode (see Copy Mode).

  If the '`use_charnames_in_special`' keyword is set in the `DESC` file, special characters no longer cause an error; they are simply output verbatim. Additionally, the backslash is represented as `\\`.

  '`use_charnames_in_special`' is currently used by `grohtml` only.

`.devicem` *xx*
\Y*n*
\Y(*nm*
\Y[*name*]
: This is approximately equivalent to '`\X'\*[`*name*`]'`' (one-character name*n*, tw o-character name*nm*). Ho wever, the contents of the string or macro*name* are not interpreted; also it is permitted for *name* to have been defined as a macro and thus contain newlines (it is not permitted for the argument to `\X` to contain newlines). The inclusion of newlines requires an extension to the Unix `troff` output format, and confuses drivers that do not know about this extension (see Device Control Commands).

See Output Devices.

5.31. Miscellaneous

This section documents parts of `gtroff` that cannot (yet) be categorized elsewhere in this manual.

`.nm` [*start* [*inc* [*space* [*indent*]]]]
`\n[.nm]`

Print line numbers. *start* is the line number of the*ne xt* output line. *inc* indicates which line numbers are printed. For example, the value 5 means to emit only line numbers that are multiples of 5; this defaults to 1. *space* is the space to be left between the number and the text; this defaults to one digit space. The fourth argument is the indentation of the line numbers, defaulting to zero. Both*space* and *indent* are given as multiples of digit spaces; they can be negative also. Without any arguments, line numbers are turned off.

`gtroff` reserves three digit spaces for the line number (which is printed right-justified) plus the amount given by *indent*; the output lines are concatenated to the line numbers, separated by *space*, and*without* reducing the line length. Depending on the value of the horizontal page offset (as set with the `po` request), line numbers that are longer than the reserved space stick out to the left, or the whole line is moved to the right.

Parameters corresponding to missing arguments are not changed; any non-digit argument (to be more precise, any argument starting with a character valid as a delimiter for identifiers) is also treated as missing.

If line numbering has been disabled with a call to `nm` without an argument, it can be reactivated with '`.nm +0`', using the previously active line numbering parameters.

The parameters of `nm` are associated with the current environment (see Environments). The current output line number is available in the register `ln`.

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*]
: Temporarily turn off line numbering. The argument is the number of lines not to be numbered; this defaults to 1.

`.mc` *glyph* [*dist*]
: Print a*margin char acter* to the right of the text.[61] The first argument is the glyph to be printed. The second argument is the distance away from the right margin. If missing, the previously set value is used; default is 10 pt). For text lines that are too long (that is, longer than the text length plus*dist*), the margin char acter is directly appended to the lines.

  With no arguments the margin character is turned off. If this occurs before a break, no margin character is printed.

  For compatibility with AT&T `troff`, a call to `mc` to set the margin character can't be undone immediately; at least one line gets a margin character. Thus

```
.ll 1i
.mc \[br]
.mc
xxx
.br
xxx
```

  produces

```
xxx        |
xxx
```

  For empty lines and lines produced by the `tl` request no margin character is emitted.

  The margin character is associated with the current environment (see Environments).

  This is quite useful for indicating text that has changed, and, in fact, there are programs available for doing this (they are called `nrchbar` and `changebar` and can be found in any '`comp.sources.unix`' archive).

```
.ll 3i
.mc |
This paragraph is highlighted with a margin
character.
.sp
Vertical space isn't marked.
.br
\&
.br
But we can fake it with '\&'.
```

[61] *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*
`\n[llx]`
`\n[lly]`
`\n[urx]`
`\n[ury]`

Retrieve the bounding box of the PostScript image found in *filename*. The file m ust 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.

5.32. `gtroff` Internals

`gtroff` processes input in three steps. One or more input characters are converted to an *input token*.[62] Then, one or more input tokens are converted to an *output node*. Finally , output nodes are converted to the intermediate output language understood by all output devices.

Actually, before step one happens, `gtroff` converts certain escape sequences into reserved input characters (not accessible by the user); such reserved characters are used for other internal processing also – this is the very reason why not all characters are valid input. See Identifiers, for more on this topic.

For example, the input string '`fi\[:u]`' is converted into a character token '`f`', a character token '`i`', and a special token '`:u`' (representing u umlaut). Later on, the character tokens '`f`' and '`i`' are merged to a single output node representing the ligature glyph '`fi`' (provided the current font has a glyph for this ligature); the same happens with '`:u`'. All output glyph nodes are 'processed', which means that they are invariably associated with a given font, font size, advance width, etc. During the formatting process, `gtroff` itself adds various nodes to control the data flow.

Macros, diversions, and strings collect elements in two chained lists: a list of input tokens that have been passed unprocessed, and a list of output nodes. Consider the following the diversion.

```
.di xxx
a
\!b
c
.br
```

[62] Except the escapes `\f`, `\F`, `\H`, `\m`, `\M`, `\R`, `\s`, and `\S`, which are processed immediately if not in copy mode.

```
.di
```

It contains these elements.

node list	token list	element number
line start node	—	1
glyph node `a`	—	2
word space node	—	3
—	`b`	4
—	`\n`	5
glyph node `c`	—	6
vertical size node	—	7
vertical size node	—	8
—	`\n`	9

Elements 1, 7, and 8 are inserted by `gtroff`; the latter two (which are always present) specify the vertical extent of the last line, possibly modified by `\x`. The`br` request finishes the current partial line, inserting a newline input token, which is subsequently converted to a space when the diversion is reread. Note that the word space node has a fixed width that isn't stretchable anymore. To convert horizontal space nodes back to input tokens, use the `unformat` request.

Macros only contain elements in the token list (and the node list is empty); diversions and strings can contain elements in both lists.

Note that the `chop` request simply reduces the number of elements in a macro, string, or diversion by one. Exceptions are*compatibility sa ve* and *compatibility ignore* input tokens, which are ignored. The `substring` request also ignores those input tokens.

Some requests like `tr` or `cflags` work on glyph identifiers only; this means that the associated glyph can be changed without destroying this association. This can be very helpful for substituting glyphs. In the following example, we assume that glyph '`foo`' isn't available by default, so we provide a substitution using the `fchar` request and map it to input character '`x`'.

```
.fchar \[foo] foo
.tr x \[foo]
```

Now let us assume that we install an additional special font '`bar`' that has glyph '`foo`'.

```
.special bar
.rchar \[foo]
```

Since glyphs defined with `fchar` are searched before glyphs in special fonts, we must call `rchar` to remove the definition of the fallback glyph. Anyway, the translation is still active; '`x`' now maps to the real glyph '`foo`'.

Macro and request arguments preserve the compatibility mode:

```
.cp 1     \" switch to compatibility mode
.de xx
\\$1
..
.cp 0     \" switch compatibility mode off
.xx caf\['e]
    ⇒ café
```

Since compatibility mode is on while `de` is called, the macro `xx` activates compatibility mode while executing. Argument`$1` can still be handled properly because it inherits the compatibility mode status which was active at the point where `xx` is called.

After expansion of the parameters, the compatibility save and restore tokens are removed.

5.33. Debugging

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 multi-stage pipeline processing. 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 v ariety 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, diversions, or boxes), environments, registers, and page location traps to the standard error stream.

`.lf` *line* [*filename*]
: 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*ne xt* 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` *string*
`.tm1` *string*
`.tmc` *string*
: Send *string*, which consumes the remainder of the input line, to the standard error stream.

  string is read in copy mode.

  The `tm` request ignores leading spaces of *string*; `tm1` handles its argument similar to the `ds` request: a leading double quote in*string* is stripped to allo w initial blanks.

  The `tmc` request is similar to `tm1` but does not append a newline (as is done in `tm` and `tm1`).

`.ab` [*string*]
: Write *string* to the standard error stream (like `tm`)and then abort GNU `troff`; that is, stop processing and terminate with a failure status. With no argument, the message written is '`User Abort.`'.

`.ex`
: 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 I/O.

When doing something involved it is useful to leave the debugging statements in the code and have them turned on by a command-line flag.

```
.if \n[DB] .tm debugging output
```

To activate 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`
: Report the contents of the current environment and all the currently defined environments (both named and numbered) to the standard error stream.

`.pm`
: 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 b y itself, the returned sizes can be larger than expected.

`.pnr`
: Report the names and contents of all currently defined registers to the standard error stream.

`.ptr`
: 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`
: Instruct `gtroff` to flush its output immediately. The intent is for interactive use, but this behaviour is currently not implemented in `gtroff`. Contrary to Unix `troff`, TTY output is sent to a device driver also (`grotty`), making it non-trivial to communicate interactively.

  This request causes a line break.

`.backtrace`
: Print a backtrace of the input stack to the standard error stream.

  Consider the following in file `test`:

```
.de xxx
.  backtrace
..
.de yyy
.  xxx
..
.
.yyy
```

  On execution, `gtroff` prints the following:

```
gtroff: backtrace: 'test':2: macro 'xxx'
```

```
gtroff: backtrace: 'test':5: macro 'yyy'
gtroff: backtrace: file 'test':8
```

The option `-b` of `gtroff` causes a backtrace to be generated on each error and warning. Warnings have to be enabled; see Warnings.

`\n[slimit]`
Use the `slimit` register to set the maximum number of objects on the input stack. If `slimit` is less than or equal to 0, there is no limit set. With no limit, a buggy recursive macro can exhaust virtual memory.

The default value is 1000; this is a compile-time constant.

`.warnscale` *si*
Set the scaling indicator used in warnings to *si*. Valid values for *si* are 'u', 'i', 'c', 'p', and 'P'. At startup, it is set to 'i'.

`.spreadwarn` [*limit*]
Emit a `break` warning if the additional space inserted for each space between words in an output line adjusted to both margins with '`.ad b`' is larger than or equal to *limit*. A negative value is treated as zero; an absent argument toggles the warning on and off without changing *limit*. The default scaling indicator is 'm'. At startup, `spreadwarn` is inactive and *limit* is 3 m.

For example,

```
.spreadwarn 0.2m
```

causes a warning if `break` warnings are not suppressed and `gtroff` must add 0.2 m or more for each interword space in a line. See Warnings.

`gtroff` has command-line options for printing out more warnings (`-w`) and for printing backtraces (`-b`) when a warning or an error occurs. The most verbose level of warnings is `-ww`.

`.warn` [*flags*]
`\n[.warn]`
Control the level of warnings checked for. The *flags* are the sum of the numbers associated with each warning that is to be enabled; all other warnings are disabled. The number associated with each warning is listed below. For example, '`.warn 0`' disables all warnings, and '`.warn 1`' disables all warnings except that about missing glyphs. If no argument is given, all warnings are enabled.

The read-only register `.warn` contains the current warning level.

5.33.1. Warnings

The warnings that can be given to `gtroff` are divided into the following categories. The name associated with each warning is used by the `-w` and `-W` options; the number is used by the `warn` request and by the `.warn` register.

'`char`'
'`1`' Non-existent glyphs.[63] This is enabled by default.

[63] `char` is a misnomer since it reports missing glyphs—there aren't missing input characters, only invalid ones.

‘number’
‘2’ Invalid numeric expressions. This is enabled by default. See Expressions.

‘break’
‘4’ In fill mode, lines that could not be broken so that their length was less than the line length. This is enabled by default.

‘delim’
‘8’ Missing or mismatched closing delimiters.

‘el’
‘16’ Use of the `el` request with no matching `ie` request. See if-else.

‘scale’
‘32’ Meaningless scaling indicators.

‘range’
‘64’ Out of range arguments.

‘syntax’
‘128’ Invalid syntax.

‘di’
‘256’ Use of `di` or `da` without an argument when there is no current diversion.

‘mac’
‘512’ An undefined string, macro, diversion, or box was used. When such an object is dereferenced, an empty object of that name is automatically created. So, in most cases, at most one warning is given for each name.

This warning is also emitted upon an attempt to move an unplanted trap (see Page Location Traps). In such cases, the unplanted macro is*not* deref erenced, so it is not created if it does not exist.

‘reg’
‘1024’ Use of undefined registers. When an undefined register is used, that register is automatically defined to have a value of 0. So, in most cases, at most one warning is given for use of a particular name.

‘tab’
‘2048’ Use of a tab character where a number was expected.

‘right-brace’
‘4096’ Use of `\}` where a number was expected.

‘missing’
‘8192’ Requests that are missing non-optional arguments.

‘input’
‘16384’ Invalid input characters.

‘escape’
‘32768’ Unrecognized escape sequences. When an unrecognized escape sequence `\`*X* is encountered, the escape character is ignored, and *X* is printed.

'space'
'65536' Missing space between a request or macro and its argument. This warning is given when an undefined name longer than two characters is encountered, and the first two characters of the name make a defined name. The request or macro is not invoked. When this warning is given, no macro is automatically defined. This is enabled by default. This warning never occurs in compatibility mode.

'font'
'131072' Non-existent fonts. This is enabled by default.

'ig'
'262144' Invalid escapes in text ignored with the `ig` request. These are conditions that are errors when they do not occur in ignored text.

'color'
'524288' Color related warnings.

'file'
'1048576' Missing files. The`mso` request giv es this warning when the requested macro file does not exist. This is enabled by default.

'all' All warnings except '`di`', '`mac`' and '`reg`'. It is intended that this covers all warnings that are useful with traditional macro packages.

'w' All warnings.

5.34. Implementation Differences

GNU `troff` has a number of features that cause incompatibilities with documents written using old versions of `troff`. Some GNU extensions to `troff` have become supported by other implementations.

GNU `troff` does not always hyphenate words as AT&T `troff` does. The AT&T implementation uses a set of hard-coded rules specific to U.S. English, while GNU `troff` uses language-specific hyphenation pattern files derived from TEX. Furthermore, in old versions of `troff` there was a limited amount of space to store hyphenation exceptions (arguments to the `hw` request); GNU `troff` has no such restriction.

Long names may be GNU `troff`'s most obvious innovation. AT&T `troff` interprets '`.dsabcd`' as defining a string '`ab`' with contents '`cd`'. Normally, GNU `troff` interprets this as a call of a macro named `dsabcd`. AT&T `troff` also interprets `\*[` and `\n[` as a reference to a string or register, respectively, called '`[`'. In GNU`troff`, ho wever, the '`[`' is normally interpreted as delimiting a long name. In compatibility mode, GNU `troff` interprets names in the traditional way, which means that they are limited to one or two characters.

`.cp` [*n*]
`.do` *name*
`\n[.C]`
`\n[.cp]`
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.

The `do` request interprets the string, request, diversion, or macro*name* (along with any further arguments) with compatibility mode disabled. Compatibility mode is restored (only if it was active) when the *expansion* of *name* is interpreted; that is, the restored compatibility state applies to the contents of the macro (string, ...)*name* as well as file or pipe data read if*name* is the `so`, `mso`, or `pso` request.

The following example illustrates several aspects of `do` behavior.

```
.de mac1
FOO
..
.de1 mac2
groff
.mac1
..
.de mac3
compatibility
.mac1
..
.de ma
\\$1
..
.cp 1
.do mac1
.do mac2 \" mac2, defined with .de1, calls "mac1"
.do mac3 \" mac3 calls "ma" with argument "c1"
.do mac3 \[ti] \" groff syntax accepted in .do arguments
    ⇒ FOO groff FOO compatibility c1 ~
```

The read-only register `.cp`, meaningful only when dereferenced from a `do` request, is 1 if compatibility mode was on when the `do` request was encountered, and 0 if it was not. This register is specialized and may require a statement of rationale.

When writing macro packages or documents that use GNU `troff` features and which may be mixed with other packages or documents that do not—common scenarios include serial processing of man pages or use of the `so` or `mso` requests—you may desire correct operation regardless of compatibility mode in the surrounding context. It may occur to you to save the existing value of '`\n(.C`' into a register, say, '`_C`', at the beginning of your file, turn compatibility mode off with '`.cp 0`', then restore it from that register at the end with '`.cp \n(_C`'. At the same time, a modular design of a document or macro package may lead you to multiple layers of inclusion. You cannot use the same register name everywhere or you risk "clobbering" the value from a preceding or enclosing context. The two-character register name space of AT&T `troff` is confining and mnemonically challenging; you may wish to use the more capacious name space of GNU `troff`. However, attempting '`.nr _my_saved_C \n(.C`' will not work in compatibility mode; the register name is too long. "This is exactly what `do` is for," you think, '`.do nr _my_saved_C \n(.C`'. The foregoing will always save zero to your register, because `do` turns compatibility mode *off* while it interprets its argument list. What you need is:

```
.do nr _my_saved_C \n[.cp]
.cp 0
```

at the beginning of your file, followed by

```
.cp _my_saved_C
```

at the end. As in the C language, we all have to share one big name space, so choose a register name that is unlikely to collide with other uses.

AT&T `troff` and other implementations handle the `lf` request differently. For them, its *line* argument changes the line number of the *current* line .

Normally, GNU `troff` preserves the input level in delimited arguments, but not in compatibility mode.

```
.ds xx '
\w'abc\*(xxdef'
    ⇒ 168 (normal mode on a terminal device)
    ⇒ 72def' (compatibility mode on a terminal device)
```

Furthermore, the escapes `\f`, `\H`, `\m`, `\M`, `\R`, `\s`, and `\S` are transparent for recognizing the beginning of a line only in compatibility mode. For example, this code produces bold output in both cases, but the text differs.

```
.de xx
Hello!
..
\fB.xx\fP
    ⇒ .xx (normal mode)
    ⇒ Hello! (compatibility mode)
```

GNU `troff` does not allow the use of the escape sequences `\|`, `\^`, `\&`, `\{`, `\}`, `\SP`, `\'`, `\‘`, `\-`, `\_`, `\!`, `\%`, and `\c` in names of strings, macros, diversions, registers, fonts, or environments; AT&T `troff` does. The `\A` escape sequence (see Identifiers) may be helpful in avoiding use of these escape sequences in names.

Normally, the syntax form `\s`*n* accepts only a single character (a digit) for *n*, consistently with other forms that originated in AT&T `troff`, like `\*`, `\$`, `\f`, `\g`, `\k`, `\n`, and `\z`. In compatibility mode only, a non-zero *n* must be in the range 4–39. Legacy documents relying upon this quirk of parsing[64] should be migrated to another `\s` form.

Fractional point sizes cause one noteworthy incompatibility. In AT&T `troff` the `ps` request ignores scale indicators and thus ‘`.ps 10u`’ sets the point size to 10 points, whereas in GNU `troff` it sets the point size to 10 scaled points. See Fractional Type Sizes.

The `pm` request differs from AT&T `troff`: GNU `troff` reports the sizes of macros, strings, and diversions in bytes and ignores an argument to report only the sum of the sizes.

Unlike AT&T `troff`, GNU `troff` does not ignore the `ss` request if the output is a terminal device; instead, the values of minimal inter-word and additional inter-sentence spacing are each rounded down to the nearest multiple of 12.

[64] The Graphic Systems C/A/T phototypesetter (the original device target for AT&T `troff`) supported only a few discrete point sizes in the range 6–36, so Ossanna contrived a special case in the parser to do what the user must have meant. Kernighan warned of this in the 1992 revision of CSTR #54 (§2.3), and more recently, McIlroy referred to it as a “living fossil”.

In GNU `troff` there is a fundamental difference between (unformatted) input characters and (formatted) output glyphs. Everything that affects how a glyph is output is stored with the glyph node; once a glyph node has been constructed, it is unaffected by any subsequent requests that are executed, including `bd`, `cs`, `tkf`, `tr`, or `fp` requests. Normally, glyphs are constructed from input characters immediately before the glyph is added to the current output line. Macros, diversions, and strings are all, in fact, the same type of object; they contain lists of input characters and glyph nodes in any combination. Special characters can be both: before being added to the output, they act as input entities; afterwards, they denote glyphs. A glyph node does not behave like an input character for the purposes of macro processing; it does not inherit any of the special properties that the input character from which it was constructed might have had. Consider the following example.

```
.di x
\\\\
.br
.di
.x
```

It prints '\\' in GNU`troff`; each pair of input bac kslashes 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,[65] 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.[66]

To store an escape sequence in a diversion that is interpreted when the diversion is reread, either use the traditional `\!` transparent output facility, or, if this is unsuitable, the new `\?` escape sequence. See Diversions and `gtroff` Internals.

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.

[65] 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.

[66] This character escape is not portable to AT&T `troff`, but is to its lineal descendant, Heirloom Doctools `troff`, as of its 060716 release (July 2006).

6. Preprocessors

This chapter describes all preprocessors that come with `groff` or which are freely available.

6.1. `geqn`

6.1.1. Invoking `geqn`

Name

eqn – format equations for GNU troff or MathML

Synopsis

eqn [**–rCNR**] [**–**d *xy*] [**–f** *F*] [**–m** *n*] [**–M** *dir*] [**–p** *n*] [**–s** *n*] [**–T** *name*] [*file* . . .]
eqn ––help
eqn –v
eqn ––version

Description

The GNU version of *eqn* is part of the *groff*(7) document formatting system. *eqn* compiles descriptions of equations embedded in *roff*(7) input files into commands that are understood by *troff*(1). Normally, it should be invoked using the **–e** option of *groff*(1). Its syntax is compatible with AT&T *eqn*, its output cannot be processed with AT&T *troff*; it must be processed with GNU *troff*. If no*file* oper ands are given on the command line, or if *file* is "**–**", the standard input stream is read. Unless the **–R** option is given, *eqn* searches for the file *eqnrc* in the directories given with the **–M** option first, then in */usr/local/lib/groff/site–tmac*, */usr/local/share/groff/site–tmac*, and finally in the standard macro directory */usr/local/share/groff/1.22.4/tmac*. If it exists, *eqn* processes it before the other input files.

Only the differences between GNU *eqn* and AT&T *eqn* are described in this document. Most of the new features of the GNU *eqn* input language are based on TEX. There are some references to the differences between TEX and GNU *eqn* below; these may safely be ignored if you do not know TEX. Three points are worth special note.

- GNU *eqn* emits Presentation MathML output when invoked with the "**–T MathML**" option.
- GNU *eqn* does not provide the functionality of *neqn*: it does not support low-resolution, typewriter-like devices (although it may work adequately for very simple input).
- GNU *eqn* sets the input token "**...**" as three periods or low dots, rather than the three centered dots of AT&T *eqn*. To get three centered dots, write **cdots** or "**cdot cdot cdot**".

Controlling delimiters

If not in compatibility mode, *eqn* recognizes

```
delim on
```

as a command to restore the delimiters which have been previously disabled with a call to "**delim off**". If delimiters haven't been specified, the call has no effect.

Automatic spacing

eqn gives each component of an equation a type, and adjusts the spacing between components using that type. Possible types are described in the table below.

`ordinary`	an ordinary character such as “1” or “*x*”
`operator`	a large operator such as “ Σ”
`binary`	a binary operator such as “+”
`relation`	a relation such as “=”
`opening`	a opening bracket such as “(”
`closing`	a closing bracket such as “)”
`punctuation`	a punctuation character such as “,”
`inner`	a subformula contained within brackets
`suppress`	a type that suppresses automatic spacing adjustment

Components of an equation get a type in one of two ways.

type *t e*
: This yields an equation component that contains *e* but that has type *t*, where *t* is one of the types mentioned above. For example, **times** is defined as follows.

```
type "binary" \(mu
```

The name of the type doesn’t have to be quoted, but quoting it protects it from macro expansion.

chartype *t text*
: Unquoted groups of characters are split up into individual characters, and the type of each character is looked up; this changes the type that is stored for each character; it says that the characters in *text* from now on have type *t*. For example,

```
chartype "punctuation" .,;:
```

would make the characters “.,;:” have type punctuation whenever they subsequently appeared in an equation. The type *t* can also be **letter** or **digit**; in these cases **chartype** changes the font type of the characters. See subsection “Fonts” below.

New primitives

big *e*
: Enlarges the expression it modifies; intended to have semantics like CSS “large”. In*troff* output, the point siz e is increased by 5; in MathML output, the expression uses

```
<mstyle mathsize='big'>
```

e1 **smallover** *e2*
: This is similar to **over**; **smallover** reduces the size of *e1* and *e2*; it also puts less vertical space between *e1* or *e2* and the fraction bar. The**o ver** primitive corresponds to the TEX **over** primitive in display styles; **smallover** corresponds to **over** in non-display styles.

vcenter *e*
: This vertically centers *e* about the math axis. The math axis is the vertical position about which characters such as “+” and “−” are centered; it is also the

vertical position used for fraction bars. For example, **sum** is defined as follows.

```
{ type "operator" vcenter size +5 \(*S }
```

vcenter is silently ignored when generating MathML.

e1 **accent** *e2*

This sets *e2* as an accent over *e1*. *e2* is assumed to be at the correct height for a lowercase letter; *e2* is moved down according to whether *e1* is taller or shorter than a lowercase letter. For example, **hat** is defined as follows.

```
accent { "^" }
```

dotdot, **dot**, **tilde**, **vec**, and **dyad** are also defined using the **accent** primitive.

e1 **uaccent** *e2*

This sets *e2* as an accent under *e1*. *e2* is assumed to be at the correct height for a character without a descender; *e2* is moved down if *e1* has a descender. **utilde** is pre-defined using **uaccent** as a tilde accent below the baseline.

split "*text***"**

This has the same effect as simply

```
text
```

but *text* is not subject to macro expansion because it is quoted; *text* is split up and the spacing between individual characters is adjusted.

nosplit *text*

This has the same effect as

```
"text"
```

but because *text* is not quoted it is subject to macro expansion; *text* is not split up and the spacing between individual characters is not adjusted.

e **opprime**

This is a variant of **prime** that acts as an operator on *e*. It produces a different result from **prime** in a case such as "**A opprime sub 1**": with **opprime** the "1" is tucked under the prime as a subscript to the "A" (as is conventional in mathematical typesetting), whereas with **prime** the "1" is a subscript to the prime character. The precedence of**opprime** is the same as that of **bar** and **under**, which is higher than that of everything except **accent** and **uaccent**. In unquoted text, a neutral apostrophe (**'**) that is not the first character on the input line is treated like **opprime**.

special *text e*

This constructs a new object from *e* using a *troff*(1) macro named *text*. When the macro is called, the string **0s** contains the output for *e*, and the number registers **0w**, **0h**, **0d**, **0skern**, and **0skew** contain the width, height, depth, subscript kern, and skew of *e*. (The*subscript k ern* of an object indicates how much a subscript on that object should be "tucked in", or placed to the left relative to a non-subscripted glyph of the same size. The*sk ew* of an object is how far to the right of the center of the object an accent over it should be placed.) The macro must modify **0s** so that it outputs the desired result with its origin at the current point, and increase the current horizontal position by the width of the object. The number registers must also be modified so that they correspond to the result.

For example, suppose you wanted a construct that “cancels” an expression by drawing a diagonal line through it.

```
.EQ
define cancel 'special Ca'
.EN
.de Ca
.  ds 0s \
\Z'\\*(0s'\
\v'\\n(0du'\
\D'l \\n(0wu —\\n(0hu-\\n(0du'\
\v'\\n(0hu'
..
```

You could then cancel an expression *e* with “**cancel {** *e* **}**”.

Here’s a more complicated construct that draws a box around an expression.

```
.EQ
define box 'special Bx'
.EN
.de Bx
.ds 0s \
\Z'\h'1n'\\*(0s'\
\Z'\
\v'\\n(0du+1n'\
\D'l \\n(0wu+2n 0'\
\D'l 0 —\\n(0hu—\\n(0du-2n'\
\D'l —\\n(0wu—2n 0'\
\D'l 0 \\n(0hu+\\n(0du+2n'\
'\
\h'\\n(0wu+2n'
.nr 0w +2n
.nr 0d +1n
.nr 0h +1n
..
```

space *n*

A positive value of the integer *n* (in hundredths of an em) sets the vertical spacing before the equation, a negative value sets the spacing after the equation, replacing the default values. This primitive provides an interface to *groff*’s **\x** escape (but with opposite sign).

This keyword has no effect if the equation is part of a *pic* picture.

Extended primitives

col *n* **{ ... }**

ccol *n* **{ ... }**

lcol *n* **{ ... }**

rcol *n* **{ ... }**

pile *n* **{ ... }**

cpile *n* **{ ... }**

lpile *n* **{ ... }**

rpile *n* **{ ... }**

The integer value *n* (in hundredths of an em) increases the vertical spacing between rows, using *groff*'s **x** escape (the value has no effect in MathML mode). Negative values are possible but have no effect. If there is more than a single value given in a matrix, the biggest one is used.

Customization

When *eqn* is generating troff markup, the appearance of equations is controlled by a large number of parameters. They have no effect when generating MathML mode, which pushes typesetting and fine motions downstream to a MathML rendering engine. These parameters can be set using the **set** command.

set *p n*

This sets parameter *p* to value *n*, where *n* is an integer. For example,

```
set x_height 45
```

says that *eqn* should assume an x height of 0.45 ems.

Possible parameters are as follows. Values are in units of hundredths of an em unless otherwise stated. These descriptions are intended to be expository rather than definitive.

minimum_size

eqn won't set anything at a smaller point size than this. The value is in points.

fat_offset

The **fat** primitive emboldens an equation by overprinting two copies of the equation horizontally offset by this amount. This parameter is not used in MathML mode; instead, fat text uses

```
<mstyle mathvariant='double-struck'>
```

over_hang

A fraction bar is longer by twice this amount than the maximum of the widths of the numerator and denominator; in other words, it overhangs the numerator and denominator by at least this amount.

accent_width

When **bar** or **under** is applied to a single character, the line is this long. Normally, **bar** or **under** produces a line whose length is the width of the object to which it applies; in the case of a single character, this tends to produce a line that looks too long.

delimiter_factor

Extensible delimiters produced with the **left** and **right** primitives have a combined height and depth of at least this many thousandths of twice the maximum amount by which the sub-equation that the delimiters enclose extends away from the axis.

delimiter_shortfall
Extensible delimiters produced with the **left** and **right** primitives have a combined height and depth not less than the difference of twice the maximum amount by which the sub-equation that the delimiters enclose extends away from the axis and this amount.

null_delimiter_space
This much horizontal space is inserted on each side of a fraction.

script_space
The width of subscripts and superscripts is increased by this amount.

thin_space
This amount of space is automatically inserted after punctuation characters.

medium_space
This amount of space is automatically inserted on either side of binary operators.

thick_space
This amount of space is automatically inserted on either side of relations.

x_height
The height of lowercase letters without ascenders such as “x”.

axis_height
The height above the baseline of the center of characters such as “+” and “−”. It is important that this value is correct for the font you are using.

default_rule_thickness
This should set to the thickness of the **[ru]** character, or the thickness of horizontal lines produced with the **D** escape sequence.

num1
The **over** command shifts up the numerator by at least this amount.

num2
The **smallover** command shifts up the numerator by at least this amount.

denom1
The **over** command shifts down the denominator by at least this amount.

denom2
The **smallover** command shifts down the denominator by at least this amount.

sup1
Normally superscripts are shifted up by at least this amount.

sup2
Superscripts within superscripts or upper limits or numerators of **smallover** fractions are shifted up by at least this amount. This is usually less than **sup1**.

sup3
Superscripts within denominators or square roots or subscripts or lower limits are shifted up by at least this amount. This is usually less than **sup2**.

sub1
Subscripts are normally shifted down by at least this amount.

sub2
When there is both a subscript and a superscript, the subscript is shifted down by at least this amount.

sup_drop
The baseline of a superscript is no more than this much amount below the top of the object on which the superscript is set.

sub_drop
The baseline of a subscript is at least this much below the bottom of the object on which the subscript is set.

big_op_spacing1
The baseline of an upper limit is at least this much above the top of the object on which the limit is set.

big_op_spacing2
The baseline of a lower limit is at least this much below the bottom of the object on which the limit is set.

big_op_spacing3
The bottom of an upper limit is at least this much above the top of the object on which the limit is set.

big_op_spacing4
The top of a lower limit is at least this much below the bottom of the object on which the limit is set.

big_op_spacing5
This much vertical space is added above and below limits.

baseline_sep
The baselines of the rows in a pile or matrix are normally this far apart. In most cases this should be equal to the sum of **num1** and **denom1**.

shift_down
The midpoint between the top baseline and the bottom baseline in a matrix or pile is shifted down by this much from the axis. In most cases this should be equal to **axis_height**.

column_sep
This much space is added between columns in a matrix.

matrix_side_sep
This much space is added at each side of a matrix.

draw_lines
If this is non-zero, lines are drawn using the **D** escape sequence, rather than with the **l** escape sequence and the **[ru]** character.

body_height
The amount by which the height of the equation exceeds this is added as extra space before the line containing the equation (using **x**). The default value is 85.

body_depth
The amount by which the depth of the equation exceeds this is added as extra space after the line containing the equation (using **x**). The default value is 35.

nroff
If this is non-zero, then **ndefine** behaves like **define** and **tdefine** is ignored, otherwise **tdefine** behaves like **define** and **ndefine** is ignored. The default value is 0. (This is typically changed to 1 by the *eqnrc* file for the **ascii**, **latin1**, **utf8**, and **cp1047** devices.)

A more precise description of the role of many of these parameters can be found in Appendix H of *The TEXbook*.

Macros

Macros can take arguments. In a macro body, **$***n* where *n* is between 1 and 9, is replaced by the *n*th argument if the macro is called with arguments; if there are fewer than *n* arguments, it is replaced by nothing. A word containing a left parenthesis where the part of the word before the left parenthesis has been defined using the **define** command is recognized as a macro call with arguments; characters following the left parenthesis up to a matching right parenthesis are treated as comma-separated arguments. Commas inside nested parentheses do not terminate an argument.

sdefine *name X anything X*
This is like the **define** command, but *name* is not recognized if called with arguments.

include "*file***"**

copy "*file***"**
Include the contents of *file* (**include** and **copy** are synonyms). Lines of*file* beginning with **.EQ** or **.EN** are ignored.

ifdef *name X anything X*
If *name* has been defined by **define** (or has been automatically defined because *name* is the output device) process *anything*; otherwise ignore *anything*. *X* can be any character not appearing in *anything*.

undef *name*
Remove definition of *name*, making it undefined. Besides the macros mentioned above, the following definitions are available: **Alpha**, **Beta**, ..., **Omega** (this is the same as **ALPHA**, **BETA**, ..., **OMEGA**), **ldots** (three dots on the baseline), and **dollar**.

Fonts

eqn normally uses at least two fonts to set an equation: an italic font for letters, and a roman font for everything else. The AT&T *eqn* **gfont** command changes the font that is used as the italic font. By default this is **I**. The font that is used as the roman font can be changed using the new **grfont** command.

grfont *f*
Set the roman font to *f*.

The **italic** primitive uses the current italic font set by **gfont**; the **roman** primitive uses the current roman font set by **grfont**. There is also a new **gbfont** command, which changes the font used by the **bold** primitive. If you only use the **roman**, **italic** and **bold** primitives to changes fonts within an equation, you can change all the fonts used by your equations just by using **gfont**, **grfont** and **gbfont** commands. You can control which characters are treated as letters (and therefore set in italics) by using the **chartype** command described above. A type of**letter** causes a char acter to be set in italic type. A type of**digit** causes a char acter to be set in roman type.

Options

−−help displays a usage message, while **−v** and **−−version** show version information; all exit afterward.

−C Recognize **.EQ** and **.EN** even when followed by a character other than space or newline, and do not handle the "**delim on**" statement specially.

−d *xy*
Specify delimiters *x* and *y* for the left and right ends, respectively, of inline equations. Any **delim** statements in the source file override this.

−f *F* This is equivalent to a "**gfont** *F*" command.

−m *n*
Set the minimum point size to *n*. *eqn* will not reduce the size of subscripts or superscripts to a smaller size than *n*.

−M *dir*
Search *dir* for *eqnrc* before the default directories.

−N Don't allow newlines within delimiters. This option allows *eqn* to recover better from missing closing delimiters.

−p *n*
This says that subscripts and superscripts should be *n* points smaller than the surrounding text. This option is deprecated. Normally, *eqn* sets subscripts and superscripts at 70% of the size of the surrounding text.

−r Only one size reduction.

−R Don't load *eqnrc*.

−s *n*
This is equivalent to a "**gsize** *n*" command. This option is deprecated.*eqn* normally sets equations at whatever the current point size is when the equation is encountered.

−T *name*
The output is for device *name*. Normally, the only effect of this is to define a macro *name* with a value of **1**; *eqnrc* uses this to provide definitions appropriate for the output device. However, if the specified device is "MathML", the output is MathML markup rather than *troff* commands, and *eqnrc* is not loaded at all. The default output device is **ps**.

Files

/usr/local/share/groff/1.22.4/tmac/eqnrc
Initialization file.

MathML Mode Limitations

MathML is designed on the assumption that it cannot know the exact physical characteristics of the media and devices on which it will be rendered. It does not support fine control of motions and sizes to the same degree *troff* does. Thus:

- *eqn* parameters have no effect on the generated MathML.
- The **special**, **up**, **down**, **fwd**, and **back** operations cannot be implemented, and yield a MathML "<merror>" message instead.
- The **vcenter** keyword is silently ignored, as centering on the math axis is the MathML default.
- Characters that *eqn* sets extra large in *troff* mode—notably the integral sign—may appear too small and need to have their "<mstyle>" wrappers adjusted by hand.

As in its *troff* mode, *eqn* in MathML mode leaves the **.EQ** and **.EN** delimiters in place for displayed equations, but emits no explicit delimiters around inline equations. They can, however, be recognized as strings that begin with "$" and end with "$" and do not cross line boundaries. See section "Bugs" below for translation limits specific to *eqn*.

Bugs

Inline equations are set at the point size that is current at the beginning of the input line.

In MathML mode, the **mark** and **lineup** features don't work. These could, in theory, be implemented with "<malignmark>" elements. In MathML mode, each digit of a numeric literal gets a separate "<mn></mn>" pair, and decimal points are tagged with "<mo></mo>". This is allowed by the specification, but inefficient.

See Also

"Typesetting Mathematics—User's Guide" (2nd edition); Computing Science Technical Report #17; Brian W. Kernighan, Lorinda L. Cherry; AT&T Bell Laboratories; 1978.

The TEXbook; Donald E. Knuth; Addison-Wesley Professional; 1984. **gr off**(1), **troff**(1), **pic**(1), **groff_font**(5)

6.2. gtbl

6.2.1. Invoking gtbl

Name

tbl – format tables for troff

Synopsis

tbl [**–C**] [*file* . . .]
tbl ––help
tbl –v
tbl ––version

Description

This manual page describes the GNU version of **tbl**, which is part of the groff document formatting system. **tb l** compiles descriptions of tables embedded within **troff** input files into commands that are understood by **troff**. Normally, it should be invoked using the **–t** option of **groff.** It is highly compatible with Unix **tbl**. The output generated by GNU **tbl** cannot be processed with Unix **troff**; it must be processed with GNU **troff**. If no files are given on the command line or a filename of **–** is given, the standard input is read.

Overview

tbl expects to find table descriptions wrapped in the **.TS** (table start) and **.TE** (table end) macros. Within each such table sections, another table can be defined by using the request **.T&** before the final command **.TE**. Each table definition has the following structure:

Global options
: This is optional. This table part can use several of these options distributed in 1 or more lines. The*global option part* m ust always be finished by a **"semi-colon ;" .**

Table format specification
: This part must be given, it is not optional. It determines the number of columns (cells) of the table. Moreover each cell is classified by being central, left adjusted, or numerical, etc. This specification can have several lines, but must be finished by a **dot .** at the end of the last line. After each cell definition, column specifiers can be appended, but that's optional.

Cells are separated by a tab character by default. That can be changed by the *global option* **tab(***c***), where** *c* is an arbitrary character.

Global options

The line immediately following the **.TS** macro may contain any of the following global options (ignoring the case of characters – Unix tbl only accepts options with all characters lowercase or all characters uppercase), separated by spaces, tabs, or commas:

allbox
: Enclose each item of the table in a box.

box Enclose the table in a box.

center
Center the table (default is left-justified). The alternative keyword name **centre** is also recognized (this is a GNU *tbl* extension).

decimalpoint(*c*)
Set the character to be recognized as the decimal point in numeric columns (GNU *tbl* only).

delim(*xy*)
Use *x* and *y* as start and end delimiters for **eqn**(1).

doublebox
Enclose the table in a double box.

doubleframe
Same as **doublebox** (GNU *tbl* only).

expand
Make the table as wide as the current line length (providing a column separation factor). Ignored if one or more 'x' column specifiers are used (see below).

In case the sum of the column widths is larger than the current line length, the column separation factor is set to zero; such tables extend into the right margin, and there is no column separation at all.

frame
Same as **box** (GNU *tbl* only).

linesize(*n*)
Set lines or rules (e.g., from **box**) in *n*-point type.

nokeep
Don't use diversions to prevent page breaks (GNU *tbl* only). Normally *tbl* attempts to prevent undesirable breaks in boxed tables by using diversions. This can sometimes interact badly with macro packages' own use of diversions—when footnotes, for example, are used.

nospaces
Ignore leading and trailing spaces in data items (GNU *tbl* only).

nowarn
Turn off warnings related to tables exceeding the current line width (GNU *tbl* only).

tab(*x*)
Use the character *x* instead of a tab to separate items in a line of input data. The global options must end with a semicolon. There might be whitespace between an option and its argument in parentheses.

Table format specification

After global options come lines describing the format of each line of the table. Each such format line describes one line of the table itself, except that the last format line (which you must end with a period) describes all remaining lines of the table. A single-key character describes each column of each line of the table. Key characters can be separated by spaces or tabs. You may run format specifications for multiple lines together on the same line by separating them with commas.

You may follow each key character with specifiers that determine the font and point size of the corresponding item, that determine column width, inter-column spacing, etc. The longest format line defines the number of columns in the table; missing format descriptors at the end of format lines are assumed to be **L**. Extra columns in the data (which have no corresponding format entry) are ignored.

The available key characters are:

a,**A** Center longest line in this column and then left-justifies all other lines in this column with respect to that centered line. The idea is to use such alphabetic subcolumns (hence the name of the key character) in combination with **L**; they are called sub-columns because **A** items are indented by 1n relative to **L** entries. Example:

```
.TS
tab(;);
ln,an.
item one;1
sub-item two;2
sub-item three;3
.T&
ln,an.
item eleven;11
sub-item twenty-two;22
sub-item thirty-three;33
.TE
```

Result:

item one	1
sub-item two	2
sub-item three	3
item eleven	11
sub-item twenty-two	22
sub-item thirty-three	33

c,**C** Center item within the column.

l,**L** Left-justify item within the column.

n,**N** Numerically justify item in the column; that is, align columns of numbers vertically at the units place. If there are one or more dots adjacent to a digit, use the rightmost one for vertical alignment. If there is no dot, use the rightmost digit for vertical alignment; otherwise, center the item within the column. Alignment can be forced to a certain position using '\&'; if there are one or more instances of this special (non-printing) character present within the data, use the leftmost one for alignment. Example:

```
.TS
n.
1
1.5
1.5.3
abcde
```

```
a\&bcde
.TE
```

Result:

```
  1
  1.5
1.5.3
 abcde
  abcde
```

If numerical entries are combined with **L** or **R** entries—this this can happen if the table format is changed with **.T&**—center the widest *number* (of the data entered under the **N** specifier regime) relative to the widest **L** or **R** entry, preserving the alignment of all numerical entries. Contrary to **A** type entries, there is no extra indentation.

Using equations (to be processed with *eqn*) within columns which use the **N** specifier is problematic in most cases due to *tbl*'s algorithm for finding the vertical alignment, as described above. Using the global**delim** option, ho wever, it is possible to make *tbl* ignore the data within *eqn* delimiters for that purpose.

r,**R** Right-justify item within the column.

s,**S** Span previous item on the left into this column. Not allowed for the first column.

^ Span down entry from previous row in this column. Not allowed for the first row.

,**-** Replace this entry with a horizontal line. Note that '' and '-' can be used for table fields only, not for column separator lines.

= Replace this entry with a double horizontal line. Note that '=' can be used for table fields only, not for column separator lines.

| The corresponding column becomes a vertical rule (if two of these are adjacent, a double vertical rule). A vertical bar to the left of the first key letter or to the right of the last one produces a line at the edge of the table.

To change the data format within a table, use the **.T&** command (at the start of a line). It is followed by format and data lines (but no global options) similar to the **.TS** request.

Column specifiers

Here are the specifiers that can appear in suffixes to column key letters (in any order):

b,**B** Short form of "**fB**" (make affected entries bold).

d,**D** Start an item that vertically spans rows, using the "**^**" column specifier or "**\^**" data item, at the bottom of its range rather than vertically centering it (GNU *tbl* only). Example:

```
.TS
tab(;) allbox;
l l
```

```
l ld
r ^
l rd.
0000;foobar
T{
1111
.br
2222
T};foo
r;
T{
3333
.br
4444
T};bar
\^;\^
.TE
```

Result:

0000	foobar
1111 2222	
r	foo
3333 4444	bar

e,**E** Make equally-spaced columns. All columns marked with this specifier get the same width; this happens after the affected column widths have been computed (this means that the largest-width value controls).

f,**F** Either of these specifiers may be followed by a font name (either one or two characters long), font number (a single digit), or long name in parentheses (this last form is a GNU *tbl* extension). A one-letter font name must be separated by one or more blanks from whatever follows.

i,**I** Short form of "**fI**" (make affected entries italic).

m,**M** Call named macro before outputting table cell text (GNU *tbl* only). Either of these specifiers may be followed by a macro name (either one or two characters long), or long name in parentheses. A one-letter macro name must be separated by one or more blanks from whatever follows. The macro which name can be specified here must be defined before creating the table. As implemented currently, this macro is only called if block input is used, that is, text between "**T{**" and "**T}**". The macro should contain only simple*roff* requests to change the text block formatting, like text adjustment, hyphenation, size, or font. The macro is called *after* other cell modifications like "**b**", "**f**", or "**v**" are output. Thus the macro can overwrite other modification specifiers.

p,**P** Followed by a number, this does a point size change for the affected fields. If signed, the current point size is incremented or decremented (using a signed multi-digit number is a GNU *tbl* extension). A point size specifier followed by a

column separation number must be separated by one or more blanks.

t,**T** Start an item vertically spanning rows at the top of its range rather than vertically centering it.

u,**U** Move the corresponding column up one half-line.

v,**V** Followed by a number, this indicates the vertical line spacing to be used in a multi-line table entry. If signed, the current vertical line spacing is incremented or decremented (using a signed number instead of a signed digit is a GNU tbl extension). A vertical line spacing specifier followed by a column separation number must be separated by one or more blanks. No effect if the corresponding table entry isn't a text block.

w,**W** Minimum column width value. Must be followed either by a *troff*(1) width expression in parentheses or a unitless integer. If no unit is given, en units are used. Also used as the default line length for included text blocks. If used multiple times to specify the width for a particular column, the last entry takes effect.

x,**X** An expanded column. After computing all column widths without an **x** specifier, use the remaining line width for this column. If there is more than one expanded column, distribute the remaining horizontal space evenly among the affected columns (this is a GNU extension). This feature has the same effect as specifying a minimum column width.

z,**Z** Ignore the corresponding column for width-calculation purposes, this is, don't use the fields but only the specifiers of this column to compute its width. A number suffix on a key character is interpreted as a column separation in en units (multiplied in proportion if the **expand** option is on – in case of overfull tables this might be zero). Default separation is 3n.

The column specifier **x** is mutually exclusive with **e** and **w** (but **e** is not mutually exclusive with **w**); if specified multiple times for a particular column, the last entry takes effect: **x** unsets both **e** and **w**, while either **e** or **w** overrides **x**.

Table data

The format lines are followed by lines containing the actual data for the table, followed finally by **.TE**. Within such data lines, items are normally separated by tab characters (or the character specified with the **tab** option). Long input lines can be broken across multiple lines if the last character on the line is '\' (which vanishes after concatenation). Note that*tb l* computes the column widths line by line, applying \w on each entry which isn't a text block. As a consequence, constructions like

```
.TS
c,l.
\s[20]MM
MMMM
.TE
```

fail; you must either say

```
.TS
cp20,lp20.
MM
MMMM
```

```
.TE
```
or
```
.TS
c,l.
\s[20]MM
\s[20]MMMM
.TE
```

A dot starting a line, followed by anything but a digit is handled as a troff command, passed through without changes. The table position is unchanged in this case. If a data line consists of only '_' or '=', a single or double line, respectively, is drawn across the table at that point; if a single item in a data line consists of only '_' or '=', then that item is replaced by a single or double line, joining its neighbors. If a data item consists only of '_' or '\=', a single or double line, respectively, is drawn across the field at that point which does not join its neighbors.

A data item consisting only of '\Rx' ('x' any character) is replaced by repetitions of character 'x' as wide as the column (not joining its neighbors). A data item consisting only of '\^' indicates that the field immediately above spans downward over this row.

Text blocks

A text block can be used to enter data as a single entry which would be too long as a simple string between tabs. It is started with 'T{' and closed with 'T}'. The former must end a line, and the latter must start a line, probably followed by other data columns (separated with tabs or the character given with the **tab** global option).

By default, the text block is formatted with the settings which were active before entering the table, possibly overridden by the **m**, **v**, and **w** tbl specifiers. If either 'w' or 'x' specifiers are not given for *all* columns of a text block span, the default length of the text block (to be more precise, the line length used to process the text block diversion) is computed as L×C/(N+1), where 'L' is the current line length, 'C' the number of columns spanned by the text block, and 'N' the total number of columns in the table. Note, however, that the actual diversion width as returned in register **\n[dl]** is used eventually as the text block width. If necessary, you can also control the text block width with a direct insertion of a **.ll** request right after 'T{'.

Miscellaneous

The number register **\n[TW]** holds the table width; it can't be used within the table itself but is defined right before calling **.TE** so that this macro can make use of it.

tbl also defines a macro **.T#** which produces the bottom and side lines of a boxed table. While**tb l** does call this macro itself at the end of the table, it can be used by macro packages to create boxes for multi-page tables by calling it within the page footer. An example of this is shown by the **–ms** macros which provide this functionality if a table starts with **.TS H** instead of the standard call to the **.TS** macro.

Interaction with eqn

tbl(1) should always be called before **eqn**(1) (**groff**(1) automatically takes care of the correct order of preprocessors).

GNU tbl enhancements

There is no limit on the number of columns in a table, nor any limit on the number of text blocks. All the lines of a table are considered in deciding column widths, not just

the first 200. Table continuation (**.T&**) lines are not restricted to the first 200 lines. Numeric and alphabetic items may appear in the same column.

Numeric and alphabetic items may span horizontally. **tbl** uses register, string, macro and diversion names beginning with the digit **3**. When using**tb l** you should avoid using any names beginning with a **3**.

GNU tbl within macros

Since **tbl** defines its own macros (right before each table) it is necessary to use an 'end-of-macro' macro. Additionally, the escape character has to be switched off. Here an example.

```
.eo
.de ATABLE ..
.TS
allbox tab(;);
cl.
\$1;\$2
.TE
...
.ec
.ATABLE A table
.ATABLE Another table
.ATABLE And "another one"
```

Note, however, that not all features of *tbl* can be wrapped into a macro because *tbl* sees the input earlier than *troff*. For example, number formatting with vertically aligned decimal points fails if those numbers are passed on as macro parameters because decimal point alignment is handled by *tbl* itself: it only sees **$1**, **$2**, etc., and therefore can't recognize the decimal point.

Options

−−help displays a usage message, while **−v** and **−−version** show version information; all exit afterward.

−C Enable compatibility mode to recognize **.TS** and **.TE** even when followed by a character other than space or newline. Leader characters (\a) are handled as interpreted.

Bugs

You should use **.TS H**/**.TH** in conjunction with a supporting macro package for *all* multi-page boxed tables. If there is no header that you wish to appear at the top of each page of the table, place the **.TH** line immediately after the format section. Do not enclose a multi-page table within keep/release macros, or divert it in any other way. A text block within a table must be able to fit on one page.

The **bp** request cannot be used to force a page-break in a multi-page table. Instead, define **BP** as follows

```
.de BP
.  ie '\\n(.z'' .bp \\$1
.  el \!.BP \\$1
..
```

and use **BP** instead of **bp**.

Using \a directly in a table to get leaders does not work (except in compatibility mode). This is correct behavior: \a is an **uninterpreted** leader. To get leaders use a real leader, either by using a control A or like this:

```
.ds a \a
.TS
tab(;);
lw(1i) l.
A\*a;B
.TE
```

A leading and/or trailing '|' in a format line, such as

```
|l r|.
```

gives output which has a 1n space between the resulting bordering vertical rule and the content of the adjacent column, as in

```
.TS
tab(#);
|l r|.
left column#right column
.TE
```

If it is desired to have zero space (so that the rule touches the content), this can be achieved by introducing extra "dummy" columns, with no content and zero separation, before and/or after, as in

```
.TS
tab(#);
r0|l r0|l.
#left column#right column#
.TE
```

The resulting "dummy" columns are invisible and have zero width; note that such columns usually don't work with terminal devices.

Simple Examples

A simple table definition follows.

```
.TS
c c c .
This is   centered
Well,     this also
.TE
```

By using **c c c**, each cell in the whole table will be centered. The separating character is here the default *tab*. The result is

```
This      is      centered
Well,     this      also
```

This definition is identical to

```
.TS
tab(@);
ccc.
This@is@centered
Well,@this@also
```

```
.TE
```

Here, the separating tab character is changed to the letter @. Moreover a title can be added and the centering directions can be changed to many other formats:

```
.TS
tab(@);
c s s
l c n .
Title
left@centers@123
another@number@75
.TE
```

The result is

	Title	
left	centers	123
another	number	75

Here **l** means *left–justified*, and **n** means *numerical*, which is here *right–justified*.

See Also

“Tbl—A Program to Format Tables”; Computing Science Technical Report #49; M. E. Lesk; AT&T Bell Laboratories; 1979.

groff(1), **troff**(1)

6.3. gpic

6.3.1. Invoking gpic

Name

pic – compile pictures for troff or TeX

Synopsis

pic [**–nCSU**] [*file* . . .]
pic –t [**–czCSU**] [*file* . . .]
pic ––help
pic –v
pic ––version

Description

This manual page describes the GNU version of **pic**, which is part of the groff document formatting system. **pic** compiles descriptions of pictures embedded within **tr off** or TEX input files into commands that are understood by TEX or **troff**. Each picture starts with a line beginning with **.PS** and ends with a line beginning with **.PE**. Anything outside of **.PS** and **.PE** is passed through without change. It is the user's responsibility to provide appropriate definitions of the **PS** and **PE** macros. When the macro package being used does not supply such definitions (for example, old versions of –ms), appropriate definitions can be obtained with **–mpic**: These will center each picture.

Options

––help displays a usage message, while **–v** and **––version** show version information; all exit afterward.

–C Recognize **.PS** and **.PE** even when followed by a character other than space or newline.

–S Safer mode; do not execute **sh** commands. This can be useful when operating on untrustworthy input (enabled by default).

–U Unsafe mode; revert the default option **–S**.

–n Don't use the groff extensions to the troff drawing commands. You should use this if you are using a postprocessor that doesn't support these extensions. The extensions are described in **groff_out**(5). The**–n** option also causes **pic** not to use zero-length lines to draw dots in troff mode.

–t TEX mode.

–c Be more compatible with *tpic*. Implies**–t**. Lines beginning with\ are not passed through transparently. Lines beginning with**.** are passed through with the initial **.** changed to \. A line beginning with**.ps** is giv en special treatment: it takes an optional integer argument specifying the line thickness (pen size) in milliinches; a missing argument restores the previous line thickness; the default line thickness is 8 milliinches. The line thickness thus specified takes effect only when a non-negative line thickness has not been specified by use of the **thickness** attribute or by setting the **linethick** variable.

–z In TEX mode draw dots using zero-length lines.

The following options supported by other versions of **pic** are ignored:

–D Draw all lines using the \D escape sequence. **pic** always does this.

–T *dev*
Generate output for the **troff** device *dev*. This is unnecessary because the **troff** output generated by **pic** is device-independent.

Usage

This section describes only the differences between GNU **pic** and the original version of **pic**. Many of these differences also apply to newer versions of Unix **pic**. A complete documentation is available in the file
/usr/local/share/doc/groff–1.22.4/pic.ms

TEX mode

TEX mode is enabled by the **–t** option. In TEX mode, **pic** will define a vbox called **\graph** for each picture. Use the**figname** command to change the name of the vbox. You must yourself print that vbox using, for example, the command

\centerline{\box\graph}
Actually, since the vbox has a height of zero (it is defined with \vtop) this will produce slightly more vertical space above the picture than below it;

\centerline{\raise 1em\box\graph}
would avoid this.

To make the vbox having a positive height and a depth of zero (as used e.g., by LATEX's **graphics.sty**), define the following macro in your document:

```
\def\gpicbox#1{%
  \vbox{\unvbox\csname #1\endcsname\kern 0pt}}
```

Now you can simply say **\gpicbox{graph}** instead of \box\graph. You must use a TEX driver that supports *tpic* version 2 specials. (*tpic* was a fork of AT&T *pic* by Tim Morgan of the University of California at Irvine that diverged from its source around 1984. It is best known today for lending its name to a group of **\special** commands it produced for TEX.)

Lines beginning with \ are passed through transparently; a **%** is added to the end of the line to avoid unwanted spaces. You can safely use this feature to change fonts or to change the value of **\baselineskip**. Anything else may well produce undesirable results; use at your own risk. Lines beginning with a period are not given any special treatment.

Commands

for *variable* **=** *expr1* **to** *expr2* [**by** [*]*expr3*] **do** *X body X*
Set *variable* to *expr1*. While the value of *variable* is less than or equal to *expr2*, do *body* and increment *variable* by *expr3*; if **by** is not given, increment *variable* by 1. If *expr3* is prefixed by * then *variable* will instead be multiplied by *expr3*. The value of *expr3* can be negative for the additive case; *variable* is then tested whether it is greater than or equal to *expr2*. For the multiplicative case, *expr3* must be greater than zero. If the constraints aren't met, the loop isn't executed. *X* can be any character not occurring in *body*.

if *expr* **then** *X if-true X* [**else** *Y if-false Y*]
Evaluate *expr*; if it is non-zero then do *if-true*, otherwise do *if-false*. *X* can be any character not occurring in *if-true*. *Y* can be any character not occurring in *if-false*.

print *arg* . . .
Concatenate the arguments and print as a line on stderr. Each*arg* m ust be an expression, a position, or text. This is useful for debugging.

command *arg* . . .
Concatenate the arguments and pass them through as a line to troff or TEX. Each *arg* must be an expression, a position, or text. This has a similar effect to a line beginning with **.** or \, but allows the values of variables to be passed through. For example,

```
.PS
x = 14
command ".ds string x is " x "."
.PE
\*[string]
```

prints

```
x is 14.
```

sh *X command X*
Pass *command* to a shell. *X* can be any character not occurring in *command*.

copy "*filename***"**
Include *filename* at this point in the file.

copy [**"***filename***"**] **thru** *X body X* [**until** "*word***"**]

copy [**"***filename***"**] **thru** *macro* [**until** "*word***"**]
This construct does *body* once for each line of *filename*; the line is split into blank-delimited words, and occurrences of **$***i* in *body*, for *i* between 1 and 9, are replaced by the *i*-th word of the line. If*filename* is not giv en, lines are taken from the current input up to **.PE**. If an**until** clause is specified, lines will be read only until a line the first word of which is *word*; that line will then be discarded. *X* can be any character not occurring in *body*. For example,

```
.PS
copy thru % circle at ($1,$2) % until "END"
1 2
3 4
5 6
END
box
.PE
```

is equivalent to

```
.PS
circle at (1,2)
circle at (3,4)
circle at (5,6)
box
```

```
.PE
```

The commands to be performed for each line can also be taken from a macro defined earlier by giving the name of the macro as the argument to **thru**. **reset**

reset *variable1*[**,**] *variable2 …*

Reset pre-defined variables *variable1*, *variable2* … to their default values. If no arguments are given, reset all pre-defined variables to their default values. Note that assigning a value to **scale** also causes all pre-defined variables that control dimensions to be reset to their default values times the new value of scale.

plot *expr* [**"***text***"**]

This is a text object which is constructed by using *text* as a format string for sprintf with an argument of *expr*. If*te xt* is omitted a format string of **"%g"** is used. Attributes can be specified in the same way as for a normal text object. Be very careful that you specify an appropriate format string; **pic** does only very limited checking of the string. This is deprecated in favour of **sprintf**.

variable **:=** *expr*

This is similar to **=** except *variable* must already be defined, and *expr* will be assigned to *variable* without creating a variable local to the current block. (By contrast, **=** defines the variable in the current block if it is not already defined there, and then changes the value in the current block only.) For example, the following:

```
.PS
x = 3
y = 3
[
  x := 5
  y = 5
]
print x " " y
.PE
```

prints

```
5 3
```

Arguments of the form

X anything X are also allo wed to be of the form

{ *anything* **}**

In this case *anything* can contain balanced occurrences of **{** and **}**. Strings may contain *X* or imbalanced occurrences of **{** and **}**.

Expressions

The syntax for expressions has been significantly extended: *x* ^ *y* (exponentiation)
sin(*x***)**
cos(*x***)**
atan2(*y***, *x***)**
log(*x***)** (base 10)
exp(*x***)** (base 10, i.e. 10^x)
sqrt(*x***)**

int(*x***)**
rand() (return a random number between 0 and 1)
rand(*x***)** (return a random number between 1 and *x*; deprecated)
srand(*x***)** (set the random number seed)
max(*e1***, *e2***)**
min(*e1***, *e2***)**
!*e*
e1 **&&** *e2*
e1 || *e2*
e1 **==** *e2*
e1 **!=** *e2*
e1 **>=** *e2*
e1 **>** *e2*
e1 **<=** *e2*
e1 **<** *e2*
"*str1***" == "***str2***"**
"*str1***" != "***str2***"**

String comparison expressions must be parenthesised in some contexts to avoid ambiguity.

Other changes

A bare expression, *expr*, is acceptable as an attribute; it is equivalent to *dir expr*, where *dir* is the current direction. For example

line 2i

means draw a line 2 inches long in the current direction. The 'i' (or 'I') character is ignored; to use another measurement unit, set the *scale* variable to an appropriate value. The maximum width and height of the picture are taken from the variables **maxpswid** and **maxpsht**. Initially these have values 8.5 and 11.

Scientific notation is allowed for numbers. For example

x = 5e−2

Text attributes can be compounded. For example,

"foo" above ljust

is valid. There is no limit to the depth to which blocks can be examined. For example,

[A: [B: [C: box]]] with .A.B.C.sw at 1,2
circle at last [].A.B.C

is acceptable.

Arcs now have compass points determined by the circle of which the arc is a part. Circles, ellipses, and arcs can be dotted or dashed. In TEX mode splines can be dotted or dashed also.

Boxes can have rounded corners. The**rad** attrib ute specifies the radius of the quarter-circles at each corner. If no**rad** or **diam** attrib ute is given, a radius of **boxrad** is used. Initially, **boxrad** has a value of 0. A box with rounded corners can be dotted or dashed. Boxes can have slanted sides. This effectively changes the shape of a box from a rectangle to an arbitrary parallelogram. The**xslanted** and **yslanted** attrib utes specify the x and y offset of the box's upper right corner from its default position.

The **.PS** line can have a second argument specifying a maximum height for the picture. If the width of zero is specified the width will be ignored in computing the scaling factor for the picture. Note that GNU**pic** will alw ays scale a picture by the same amount vertically as well as horizontally. This is different from the DWB 2.0 **pic** which may scale a picture by a different amount vertically than horizontally if a height is specified. Each text object has an invisible box associated with it. The compass points of a text object are determined by this box. The implicit motion associated with the object is also determined by this box. The dimensions of this box are taken from the width and height attributes; if the width attribute is not supplied then the width will be taken to be **textwid**; if the height attribute is not supplied then the height will be taken to be the number of text strings associated with the object times **textht**. Initially **textwid** and **textht** have a value of 0.

In (almost all) places where a quoted text string can be used, an expression of the form

> **sprintf("***format***",** *arg***,...)** can also be used; this will produce the arguments formatted according to *format*, which should be a string as described in **printf**(3) appropriate for the number of arguments supplied. Only the flags '**#**', '**–**', '', and ' ' (space), a minimum field width, an optional precision, and the conversion specifications **%e**, **%E**, **%f**, **%g**, **%G**, and **%%** are supported.

The thickness of the lines used to draw objects is controlled by the **linethick** variable. This gives the thickness of lines in points. A negative value means use the default thickness: in TEX output mode, this means use a thickness of 8 milliinches; in TEX output mode with the **–c** option, this means use the line thickness specified by **.ps** lines; in troff output mode, this means use a thickness proportional to the pointsize. A zero value means draw the thinnest possible line supported by the output device. Initially it has a value of –1. There is also a **thick**[**ness**] attribute. For example,

circle thickness 1.5

would draw a circle using a line with a thickness of 1.5 points. The thickness of lines is not affected by the value of the **scale** variable, nor by the width or height given in the **.PS** line. Boxes (including boxes with rounded corners or slanted sides), circles and ellipses can be filled by giving them an attribute of **fill**[**ed**]. This takes an optional argument of an expression with a value between 0 and 1; 0 will fill it with white, 1 with black, values in between with a proportionally gray shade. A value greater than 1 can also be used: this means fill with the shade of gray that is currently being used for text and lines. Normally this will be black, but output devices may provide a mechanism for changing this. Without an argument, then the value of the variable **fillval** will be used. Initially this has a value of 0.5. The invisible attribute does not affect the filling of objects. Any text associated with a filled object will be added after the object has been filled, so that the text will not be obscured by the filling.

Three additional modifiers are available to specify colored objects: **outline**[**d**] sets the color of the outline, **shaded** the fill color, and **colo**[**u**]**r**[**ed**] sets both. All three keywords expect a suffix specifying the color, for example

circle shaded "green" outline "black"

Currently, color support isn't available in TEX mode. Predefined color names for **groff** are in the device macro files, for example **ps.tmac**; additional colors can be defined with the **.defcolor** request (see the manual page of **troff**(1) for more details). To change the name of the vbox in TEX mode, set the pseudo-variable **figname** (which

is actually a specially parsed command) within a picture. Example:

```
.PS
figname = foobar;
...
.PE
```

The picture is then available in the box **foobar**.

pic assumes that at the beginning of a picture both glyph and fill color are set to the default value. Arrow heads will be drawn as solid triangles if the variable **arrowhead** is non-zero and either TEX mode is enabled or the **–n** option has not been given. Initially **arrowhead** has a value of 1. Note that solid arrow heads are always filled with the current outline color.

The troff output of **pic** is device-independent. The**–T** option is theref ore redundant. All numbers are taken to be in inches; numbers are never interpreted to be in troff machine units. Objects can have an **aligned** attribute. This will only work if the post-processor is **grops**, or **gropdf**. Any text associated with an object having the **aligned** attribute will be rotated about the center of the object so that it is aligned in the direction from the start point to the end point of the object. Note that this attribute will have no effect for objects whose start and end points are coincident.

In places where *n***th** is allowed '*expr*'**th** is also allowed. Note that'**th** is a single token: no space is allowed between the ' and the **th**. For example,

```
for i = 1 to 4 do {
  line from 'i'th box.nw to 'i+1'th box.se
}
```

Conversion

To obtain a stand-alone picture from a **pic** file, enclose your **pic** code with **.PS** and **.PE** requests; **roff** configuration commands may be added at the beginning of the file, but no **roff** text. It is necessary to feed this file into **groff** without adding any page information, so you must check which **.PS** and **.PE** requests are actually called. For example, the mm macro package adds a page number, which is very annoying. At the moment, calling standard **groff** without any macro package works. Alternatively, you can define your own requests, e.g., to do nothing:

```
.de PS
..
.de PE
..
```

groff itself does not provide direct conversion into other graphics file formats. But there are lots of possibilities if you first transform your picture into PostScript® format using the **groff** option **–Tps**. Since this*ps*-file lac ks BoundingBox information it is not very useful by itself, but it may be fed into other conversion programs, usually named **ps2***other* or **psto***other* or the like. Moreover, the PostScript interpreter **ghostscript** (**gs**) has built-in graphics conversion devices that are called with the option

gs –sDEVICE=*<devname>*

Call

gs −−help
for a list of the available devices.

An alternative may be to use the **−Tpdf** option to convert your picture directly into **PDF** format. The MediaBox of the file produced can be controlled by passing a **−P−p** papersize to groff. As the Encapsulated PostScript File Format **EPS** is getting more and more important, and the conversion wasn't regarded trivial in the past you might be interested to know that there is a conversion tool named **ps2eps** which does the right job. It is much better than the tool **ps2epsi** packaged with **gs**.

For bitmapped graphic formats, you should use **pstopnm**; the resulting (intermediate) **PNM** file can be then converted to virtually any graphics format using the tools of the **netpbm** package.

Files

/usr/local/share/groff/1.22.4/tmac/pic.tmac
Example definitions of the **PS** and **PE** macros.

Bugs

Characters that are invalid as input to GNU *troff* (see the *groff* Texinfo manual or *groff_char*(7) for a list) are rejected even in TEX mode. The interpretation of**fillv al** is incompatible with the *pic* in Tenth Edition Research Unix, which interprets 0 as black and 1 as white.

See Also

/usr/local/share/doc/groff−1.22.4/pic.ps
"Making Pictures with GNU pic"; Eric S. Raymond. This file, together with its source, *pic.ms*, is part of the *groff* distribution.

"PIC—A Graphics Language for Typesetting: User Manual"; Computing Science Technical Report #116; Brian W. Kernighan; AT&T Bell Laboratories; 1991. **ps2eps** is available from CTAN mirrors, e.g., ftp://ftp.dante.de/tex-archive/support/ps2eps/

W. Richard Stevens, *Turning PIC into HTML* W. Richard Stevens, *"Examples of pic Macros"*

troff(1), **groff_out**(5), **tex**(1), **gs**(1), **ps2eps**(1), **pstopnm**(1), **ps2epsi**(1), **pnm**(5)

6.3.2. Using gpic

Making Pictures With GNU PIC

Eric S. Raymond

⟨*esr@snark.thyrsus.com*⟩

The **pic** language is a **troff** extension that makes it easy
to create and alter box-and-arrow diagrams of the kind frequently used
in technical papers and textbooks.
This paper is both an introduction to and reference for *gpic*(1),
the implementation distributed by the Free Software Foundation for use
with *groff*(1).
It also catalogs other implementations and explains the differences
among them.

6.3.3. Introduction to PIC

6.3.3.1. Why PIC?

The **pic** language provides an easy way to write procedural box-and-arrow diagrams to be included in **troff** documents. The language is sufficiently flexible to be quite useful for state charts, Petri-net diagrams, flow charts, simple circuit schematics, jumper layouts, and other kinds of illustration involving repetitive uses of simple geometric forms and splines. Because these descriptions are procedural and object-based, they are both compact and easy to modify.

The phrase "GNU pic" may refer to either of two **pic** implementations distributed by the Free Software Foundation and intended to accept the same input language. The*gpic*(1) implementation is f or use with the *groff*(1) implementation of **troff**. The*pic2plot*(1) implementation runs standalone and is part of the **plotutils** pac kage. Because both implementations are widely available in source form for free, they are good bets for writing very portable documentation.

6.3.3.2. PIC Versions

The original 1984 pre-*ditroff*(1) version of **pic** is long obsolete. The rewritten 1991 version is still available as part of the Documenter's Work Bench module of System V.

Where differences between Documenter's Work Bench (1991) **pic** and GNU **pic** need to be described, original **pic** is referred to as "DWB pic". Details on the history of the program are given at the end of this document.

The **pic2plot** program does not require the rest of the *groff*(1) toolchain to render graphics. It can display **pic** diagrams in an X window, or generate output plots in a large number of other formats. These formats include: PNG, PBM, PGM, PPM, GIF, SVG, Adobe Illustrator format, idraw-editable Postscript, the WebCGM format for Web-based vector graphics, the format used by the **xfig** drawing editor, the Hewlett-Packard PCL 5 printer language, the Hewlett-Packard Graphics Language (by default, HP-GL/2), the ReGIS (remote graphics instruction set) format developed by DEC, Tektronix format, and device-independent GNU graphics metafile format.

In this document, *gpic*(1) and *pic2plot*(1) extensions are marked as such.

6.3.4. Invoking PIC

Every **pic** description is a little program describing drawing actions. The**[gtn]r off**-dependent versions compile the program by *pic*(1) into *gtroff*(1) macros; the *pic2plot*(1) implementation uses a plotting library to draw the picture directly. Programs that process or display *gtroff*(1) output need not know or care that parts of the image began life as **pic** descriptions.

The *pic*(1) program tries to translate anything between **.PS** and **.PE** markers, and passes through everything else. The normal definitions of**.PS** and **.PE** in the *ms* macro pac kage and elsewhere have also the side-effect of centering the **pic** output on the page.

6.3.4.1. PIC Error Messages

If you make a **pic** syntax error, *gpic*(1) issues an error message in the standard*gcc*(1)-lik e syntax. A typical error message looks like this

```
pic:pic.ms:<nnn>: parse error before `<token>'
pic:pic.ms:<nnn>: giving up on this picture
```

where ⟨nnn⟩ is a line number, and ⟨token⟩ is a token near (usually just after) the error location.

6.3.5. Basic PIC Concepts

Pictures are described procedurally, as collections of objects connected by motions. Normally, **pic** tries to string together objects left-to-right in the sequence they are described, joining them at visually natural points. Here is an example illustrating the flow of data in **pic** processing:

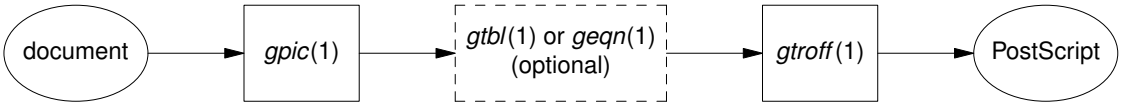


Figure 6-1: Flow of **pic** data

This was produced from the following **pic** program:

```
.PS
ellipse "document";
arrow;
box width 0.6 "\fIgpic\/\fP(1)"
arrow;
box width 1.1 "\fIgtbl\/\fP(1) or \fIgeqn\/\fP(1)" "(optional)" dashed;
arrow;
box width 0.6 "\fIgtroff\/\fP(1)";
arrow;
ellipse "PostScript"
.PE
```

This little program illustrates several **pic** basics. Firstly, we see how to invoke three object types; ellipses, arrows, and boxes. We see how to declare text lines to go within an object (and that text can have font changes in it). We see how to change the line style of an object from solid to dashed. And we see that a box can be made wider than its default size to accommodate more text (we'll discuss this facility in detail in the next section).

We also get to see **pic**'s simple syntax. Statements are ended by newlines or semicolons. String quotes are required around all text arguments, whether or not they contain spaces. In general, the order of command arguments and modifiers like "width 1.2" or "dashed" doesn't matter, except that the order of text arguments is significant.

Here are all but one of the basic **pic** objects at their default sizes:

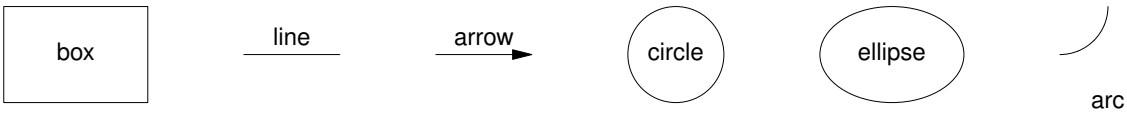


Figure 6-2: Basic **pic** objects

The missing simple object type is a *spline*. There is also a way to collect objects into *block composites* which allows you to treat the whole group as a single object (resembling a box) for many purposes. We'll describe both of these later on.

The box, ellipse, circle, and block composite objects are*closed*; lines , arrows, arcs and splines are *open*. This distinction is often important in explaining command modifiers.

Figure 6-2 was produced by the following **pic** program, which introduces some more basic concepts:

```
.PS
box "box";
move;
line "line" "";
move;
arrow "arrow" "";
move;
circle "circle";
move;
ellipse "ellipse";
move;
arc; down; move; "arc"
.PE
```

The first thing to notice is the *move* command, which moves a default distance (1/2 inch) in the current movement direction.

Secondly, see how we can also decorate lines and arrows with text. The line and arrow commands each take two arguments here, specifying text to go above and below the object. If you wonder why one argument would not do, contemplate the output of **arrow "ow!"**:

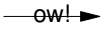


Figure 6-3: Text centered on an arrow

When a command takes one text string, **pic** tries to place it at the object's geometric center. As you add more strings, **pic** treats them as a vertical block to be centered. The program

```
line "1";
line "1" "2";
line "1" "2" "3";
line "1" "2" "3" "4";
line "1" "2" "3" "4" "5";
```

for example, gives you this:

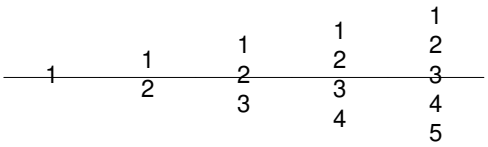


Figure 6-4: Effects of multiple text arguments

The last line of Figure 3-2's program, '**arc; down; move; "arc"**', describing the captioned arc, introduces several new ideas. Firstly, we see how to change the direction in which objects are joined. Had we written **arc; move; "arc"**, omitting **down** the caption would have been joined to the top of the arc, like this:

Figure 6-5: Result of **arc; move;**

This is because drawing an arc changes the default direction to the one its exit end points at. To reinforce this point, consider:

arc

Figure 6-6: Result of **arc cw; move;**

All we've done differently here is specify "cw" for a clockwise arc ("ccw" specifies counter-clockwise direction). Observe how it changes the default direction to down, rather than up.

Another good way to see this via with the following program:

```
line; arc; arc cw; line
```

which yields:

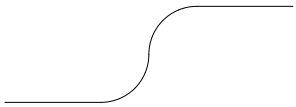


Figure 6-7: Result of **line; arc; arc cw; line**

Notice that we did not have to specify "up" for the second arc to be joined to the end of the first.

Finally, observe that a string, alone, is treated as text to be surrounded by an invisible box of a size either specified by width and height attributes or by the defaults **textwid** and **textht**. Both are initially zero (because we don't know the default font size).

6.3.6. Sizes and Spacing

Sizes are specified in inches. If you don't like inches, it's possible to set a global style variable **scale** that changes the unit. Setting**scale = 2.54** eff ectively changes the internal unit to centimeters (all other size variable values are scaled correspondingly).

6.3.6.1. Default Sizes of Objects

Here are the default sizes for **pic** objects:

Object	Default Size
box	0.75" wide by 0.5" high
circle	0.5" diameter
ellipse	0.75" wide by 0.5" high
arc	0.5" radius
line	0.5" long
arrow	0.5" long

The simplest way to think about these defaults is that they make the other basic objects fit snugly into a default-sized box.

pic2plot(1) does not necessarily emit a physical inch for each virtual inch in its drawing coordinate system. Instead, it draws on a canvas 8 virtual inches by 8 virtual inches wide. If its output page size is "letter", these virtual inches will map to real ones. Specifying a different page size (such as, say, "a4") will scale virtual inches so they are output as one eighth of the page width. Also, *pic2plot*(1) centers all images by default, though the **–n** option can be used to prevent this.

6.3.6.2. Objects Do Not Stretch!

Text is rendered in the current font with normal troff line spacing. Boxes, circles, and ellipses do *not* automatically resize to fit enclosed text. Thus, if you say **box "this text far too long for a default box"** you'll get this:

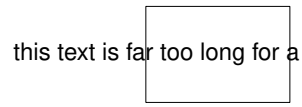


Figure 6-1: Boxes do not automatically resize

which is probably not the effect you want.

6.3.6.3. Resizing Boxes

To change the box size, you can specify a box width with the “width” modifier:

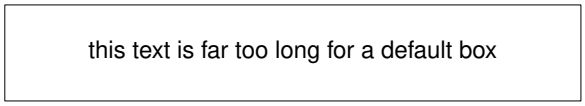


Figure 6-2: Result of **box width 3**

This modifier takes a dimension in inches. There is also a “height” modifier that changes a box’s height. The **width** keyword may be abbreviated to **wid**; the **height** keyword to **ht**.

6.3.6.4. Resizing Other Object Types

To change the size of a circle, give it a **rad[ius]** or **diam[eter]** modifier; this changes the radius or diameter of the circle, according to the numeric argument that follows.

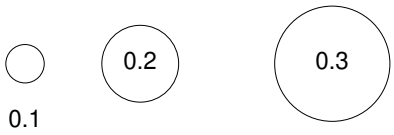


Figure 6-3: Circles with increasing radii

The **move** command can also take a dimension, which just tells it how many inches to move in the current direction.

Ellipses are sized to fit in the rectangular box defined by their axes, and can be resized with **width** and **height** like boxes.

You can also change the radius of curvature of an arc with **rad[ius]** (which specifies the radius of the circle of which the arc is a segment). Larger values yield flatter arcs.

Figure 6-4: **arc rad** with increasing radii

Observe that because an arc is defined as a quarter circle, increasing the radius also increases the size of the arc’s bounding box.

6.3.6.5. The ‘same’ Keyword

In place of a dimension specification, you can use the keyword **same**. This gives the object the same size as the previous one of its type. As an example, the program

```
.PS
box; box wid 1 ht 1; box same; box
.PE
```

gives you

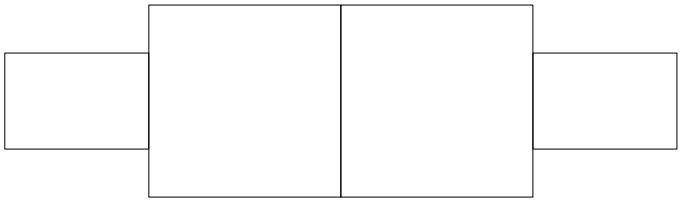


Figure 6-5: The **same** keyword

6.3.7. Generalized Lines and Splines

6.3.7.1. Diagonal Lines

It is possible to specify diagonal lines or arrows by adding multiple **up**, **down**, **left**, and **right** modifiers to the line object. Any of these can have a multiplier. To understand the effects, think of the drawing area as being gridded with standard-sized boxes.

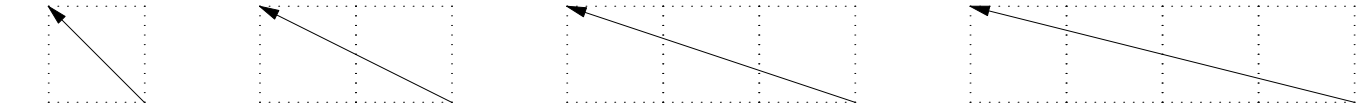


Figure 6-1: Diagonal arrows (dotted boxes show the implied 0.5-inch grid)

6.3.7.2. Multi-Segment Line Objects

A “line” or “arrow” object may actually be a path consisting of any number of segments of varying lengths and directions. To describe a path, connect several line or arrow commands with the keyword **then**.

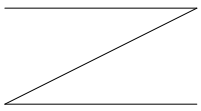


Figure 6-2: **line right 1 then down .5 left 1 then right 1**

If a path starts with **then**, the first segment is assumed to be into the current direction, using the default length.

6.3.7.3. Spline Objects

If you start a path with the **spline** keyword, the path vertices are treated as control points for a spline curve fit.

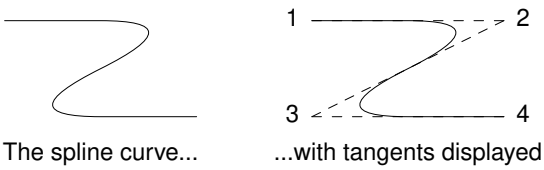


Figure 6-3: **spline right 1 then down .5 left 1 then right 1**

You can describe many natural-looking but irregular curves this way. For example:

spline right then up then left then down ->; **spline left then up right then down right ->;**

Figure 6-4: Two more spline examples

Note the arrow decorations. Arrowheads can be applied naturally to any path-based object, line or spline. We'll see how in the next section.

6.3.8. Decorating Objects

6.3.8.1. Text Special Effects

All **pic** implementations support the following font-styling escapes within text objects:

\fR, \f1
: Set Roman style (the default)

\fI, \f2 Set Italic style

\fB, \f3
: Set Bold style

\fP
: Revert to previous style; only works one level deep, does not stack.

In the **pic** implementations that are preprocessors for a toolchain that include **[gtn]roff**, text objects may also contain **[gtn]roff** vertical- and horizontal-motion escapes such as \h or \v. Troff special glyphs are also available. All \-escapes will be passed through to the postprocessing stage and have their normal effects. The base font family is set by the **[gtn]roff** environment at the time the picture is rendered.

pic2plot replaces **[gtn]roff** horizontal- and vertical-motion escapes with \-escapes of its own. Troff special glyphs are not available, but in most back ends Latin-1 special characters and a square-root radical will be. See the**pic2plot** documentation for full details.

6.3.8.2. Dashed Objects

We've already seen that the modifier **dashed** can change the line style of an object from solid to dashed. GNU **gpic** permits you to dot or dash ellipses, circles, and arcs (and splines in TEX mode only); some versions of DWB may only permit dashing of lines and boxes. It's possible to change the dash interval by specifying a number after the modifier.

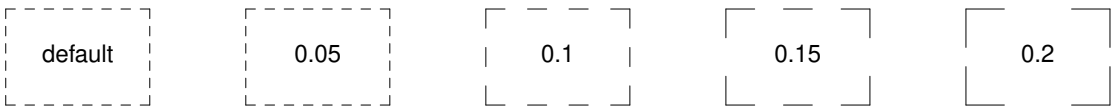


Figure 6-1: Dashed objects

6.3.8.3. Dotted Objects

Another available qualifier is **dotted**. GNU**gpic** permits y ou to dot or dash ellipses, circles, and arcs (and splines in TEX mode only); some versions of DWB may only permit dashing of lines and boxes. It too can be suffixed with a number to specify the interval between dots:

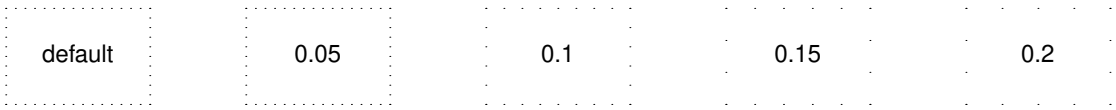


Figure 6-2: Dotted objects

6.3.8.4. Rounding Box Corners

It is also possible, in GNU **gpic** only, to modify a box so it has rounded corners:

Figure 6-3: **box rad** with increasing radius values

Radius values higher than half the minimum box dimension are silently truncated to that value.

6.3.8.5. Slanted Boxes

GNU **gpic** supports slanted boxes:

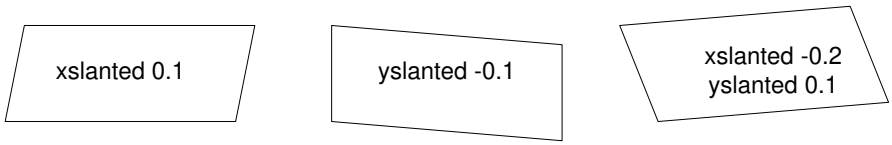


Figure 6-4: Various slanted boxes.

The **xslanted** and **yslanted** attributes specify the x and y offset, respectively, of the box's upper right corner from its default position.

6.3.8.6. Arrowheads

Lines and arcs can be decorated as well. Any line or arc (and any spline as well) can be decorated with arrowheads by adding one or more as modifiers:

Figure 6-5: Double-headed line made with **line <- ->**

In fact, the **arrow** command is just shorthand for **line ->**. And there is a double-head modifier <->, so the figure above could have been made with **line <->**.

Arrowheads have a **width** attribute, the distance across the rear; and a **height** attribute, the length of the arrowhead along the shaft.

Arrowhead style is controlled by the style variable **arrowhead**. The DWB and GNU versions interpret it differently. DWB defaults to open arrowheads and an **arrowhead** value of 2; the Kernighan paper says a value of 7 makes solid arrowheads. GNU**gpic** def aults to solid arrowheads and an **arrowhead** value of 1; a value of 0 produces open arrowheads. Note that solid arrowheads are always filled with the current outline color.

6.3.8.7. Line Thickness

It's also possible to change the line thickness of an object (this is a GNU extension, DWB **pic** doesn't support it). The default thickness of the lines used to draw objects is controlled by the **linethick** variable. This gives the thickness of lines in points. A negative value means use the default thickness: in TEX output mode, this means use a thickness of 8 milliinches; in TEX output mode with the **-c** option, this means use the line thickness specified by **.ps** lines; in troff output mode, this means use a thickness proportional to the pointsize. A zero value means draw the thinnest possible line supported by the output device. Initially it has a value of -1. There is also a **thickness** attribute (which can be abbreviated to

thick). For example, **circle thickness 1.5** would draw a circle using a line with a thickness of 1.5 points. The thickness of lines is not affected by the value of the **scale** variable, nor by any width or height given in the **.PS** line.

6.3.8.8. Invisible Objects

The modifier **invis[ible]** makes an object entirely invisible. This used to be useful for positioning text in an invisible object that is properly joined to neighboring ones. Newer DWB versions and GNU **pic** treat stand-alone text in exactly this way.

6.3.8.9. Filled Objects

It is possible to fill boxes, circles, and ellipses. The modifier**fill[ed]** accomplishes this . You can suffix it with a fill value; the default is given by the style variable **fillval**.

DWB **pic** and **gpic** have opposite conventions for fill values and different defaults. DWB **fillval** defaults to 0.3 and smaller values are darker; GNU **fillval** uses 0 for white and 1 for black.

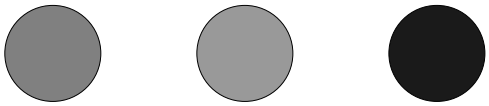


Figure 6-6: **circle fill; move; circle fill 0.4; move; circle fill 0.9;**

GNU **gpic** makes some additional guarantees. A fill value greater than 1 can also be used: this means fill with the shade of gray that is currently being used for text and lines. Normally this is black, but output devices may provide a mechanism for changing this. The invisible attribute does not affect the filling of objects. Any text associated with a filled object is added after the object has been filled, so that the text is not obscured by the filling.

The closed-object modifier **solid** is equivalent to **fill** with the darkest fill value (DWB **pic** had this capability but mentioned it only in a reference section).

6.3.8.10. Colored Objects

As a GNU extension, three additional modifiers are available to specify colored objects. **outline** sets the color of the outline, **shaded** the fill color, and **color** sets both. All three keywords expect a suffix specifying the color. Example:

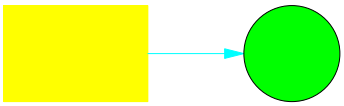


Figure 6-7: **box color "yellow"; arrow color "cyan"; circle shaded "green" outline "black";**

Alternative spellings are **colour**, **colored**, **coloured**, and **outlined**.

Predefined color names for *[gtn]roff*-based **pic** implementations are defined in the device macro files, for example `ps.tmac`; additional colors can be defined with the**.defcolor** request (see the man ual page of GNU *troff*(1) for more details). Currently, color support is not available at all in TEX mode.

The *pic2plot*(1) carries with its own set of color names, essentially those recognized by the X window system with “grey” accepted as a variant of “gray”.

pic assumes that at the beginning of a picture both glyph and fill color are set to the default value.

6.3.9. More About Text Placement

By default, text is centered at the geometric center of the object it is associated with. The modifier**ljust** causes the left end to be at the specified point (which means that the text lies to the right of the specified place!), the modifier **rjust** puts the right end at the place. The modifiers**abo ve** and **below** center the text one half line space in the given direction.

Text attributes can be combined:

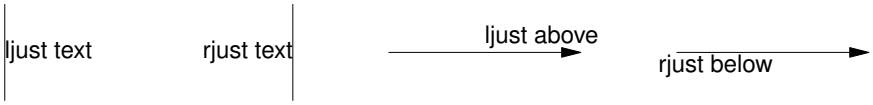


Figure 6-1: Text attributes

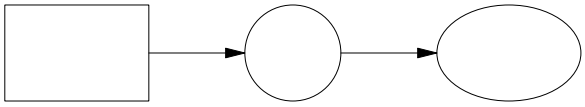
What actually happens is that *n* text strings are centered in a box that is **textwid** wide by **textht** high. Both these variables are initially zero (that is **pic**'s way of not making assumptions about *[tg]roff* (1)'s default point size).

In GNU **gpic**, objects can have an **aligned** attribute. This only works if the postprocessor is **grops** or **gropdf**. Any text associated with an object having the **aligned** attribute is rotated about the center of the object so that it is aligned in the direction from the start point to the end point of the object. Note that this attribute has no effect for objects whose start and end points are coincident.

6.3.10. More About Direction Changes

We've already seen how to change the direction in which objects are composed from rightwards to downwards. Here are some more illustrative examples:

right; box; arrow; circle; arrow; ellipse



left; box; arrow; circle; arrow; ellipse

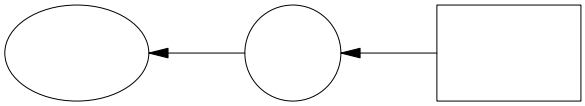
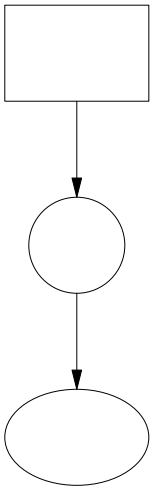


Figure 6-1: Effects of different motion directions (right and left)

down; box; arrow; circle; arrow; ellipse;



up; box; arrow; circle; arrow; ellipse;

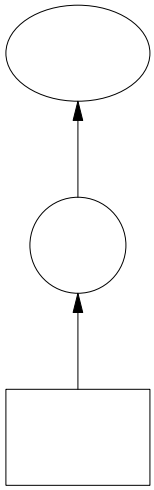


Figure 6-2: Effects of different motion directions (up and down)

Something that may appear surprising happens if you change directions in the obvious way:

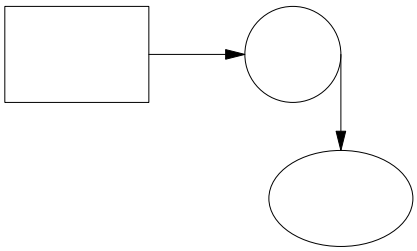


Figure 6-3: **box; arrow; circle; down; arrow; ellipse**

You might have expected that program to yield this:

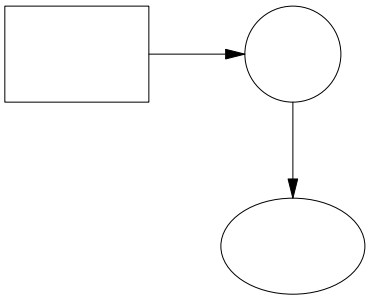


Figure 6-4: More intuitive?

But, in fact, to get Figure 6.3.10.3 you have to do this:

```
.PS
box;
arrow;
circle;
move to last circle .s;
down;
arrow;
ellipse
.PE
```

Why is this? Because the exit point for the current direction is already set when you draw the object. The second arrow in Figure 6.3.10.2 dropped downwards from the circle's attachment point for an object to be joined to the right.

The meaning of the command **move to last circle .s** should be obvious. In order to see how it generalizes, we'll need to go into detail on two important topics; locations and object names.

6.3.11. Naming Objects

The most natural way to name locations in **pic** is relative to objects. In order to do this, you have to be able to name objects. The**pic** language has rich f acilities for this that try to emulate the syntax of English.

6.3.11.1. Naming Objects By Order Of Drawing

The simplest (and generally the most useful) way to name an object is with a **last** clause. It needs to be followed by an object type name; **box**, **circle**, **ellipse**, **line**, **arrow**, **spline**, **""**, or **[]** (the last type refers to a *composite object* which we'll discuss later). So, for example, the **last circle** clause in the program attached to Figure 6.3.11.1.3 refers to the last circle drawn.

More generally, objects of a given type are implicitly numbered (starting from 1). You can refer to (say) the third ellipse in the current picture with **3rd ellipse**, or to the first box as **1st box**, or to the fifth text string (which isn't an attribute to another object) as **5th ""**.

Objects are also numbered backwards by type from the last one. You can say **2nd last box** to get the second-to-last box, or **3rd last ellipse** to get the third-to-last ellipse.

In places where *n***th** is allowed, ‘*expr*’**th** is also allowed. Note that’**th** is a single tok en: no space is allowed between the ’ and the **th**. For example,

```
for i = 1 to 4 do {
   line from `i'th box.nw to `i+1'th box.se
}
```

6.3.11.2. Naming Objects With Labels

You can also specify an object by referring to a label. A label is a word (which must begin with a capital letter) followed by a colon; you declare it by placing it immediately before the object drawing command. For example, the program

```
.PS
A: box "first" "object"
move;
B: ellipse "second" "object"
move;
arrow right at A .r;
.PE
```

declares labels **A** and **B** for its first and second objects. Here’s what that looks like:

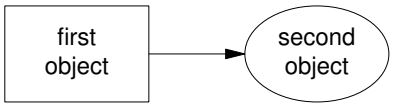


Figure 6-1: Example of label use

The **at** statement in the fourth line uses the label **A** (the behavior of **at** is explained in the next section). We’ll see later on that labels are most useful for referring to block composite objects.

Labels are not constants but variables (you can view colon as a sort of assignment). You can say something like **A: A + (1,0);** and the effect is to reassign the label **A** to designate a position one inch to the right of its old value.

6.3.12. Describing locations

The location of points can be described in many different ways. All these forms are interchangeable as for as the **pic** language syntax is concerned; where you can use one, any of the others that would make semantic sense are allowed.

The special label **Here** always refers to the current position.

6.3.12.1. Absolute Coordinates

The simplest is absolute coordinates in inches; **pic** uses a Cartesian system with (0,0) at the lower left corner of the virtual drawing surface for each picture (that is, X increases to the right and Y increases upwards). An absolute location may always be written in the conventional form as two comma-separated numbers surrounded by parentheses (and this is recommended for clarity). In contexts where it creates no ambiguity, the pair of X and Y coordinates suffices without parentheses.

It is a good idea to avoid absolute coordinates, however. They tend to make picture descriptions difficult to understand and modify. Instead, there are quite a number of ways to specify locations relative to **pic** objects and previous locations.

Another possibility of surprise is the fact that **pic** crops the picture to the smallest bounding box before writing it out. For example, if you have a picture consisting of a small box with its lower left corner at (2,2) and another small box with its upper right corner at (5,5), the width and height of the image are both 3 units and not 5. To get the origin at (0,0) included, simply add an invisible object to the picture, positioning the object’s left corner at (0,0).

6.3.12.2. Locations Relative to Objects

The symbol **Here** always refers to the position of the last object drawn or the destination of the last **move**.

Alone and unqualified, a **last circle** or any other way of specifying a closed-object or arc location refers as a position to the geometric center of the object. Unqualified, the name of a line or spline object refers to the position of the object start.

Also, **pic** objects have quite a few named locations associated with them. One of these is the object center, which can be indicated (redundantly) with the suffix **.center** (or just **.c**). Thus, **last circle .center** is equivalent to **last circle**.

6.3.12.2.1. Locations Relative to Closed Objects

Every closed object (box, circle, ellipse, or block composite) also has eight compass points associated with it;

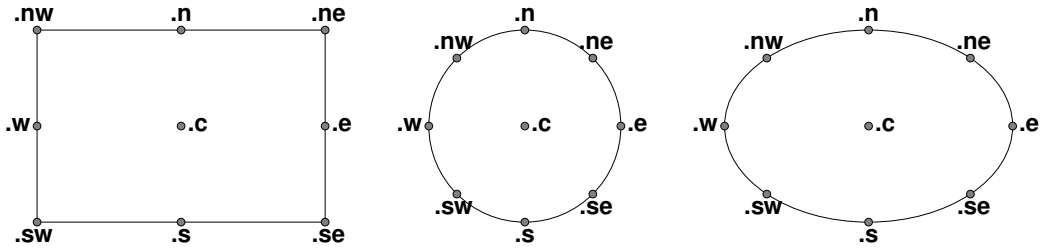


Figure 6-1: Compass points

these are the locations where eight compass rays from the geometric center would intersect the figure. So when we say **last circle .s** we are referring to the south compass point of the last circle drawn. The explanation of Figure 8-3's program is now complete.

(In case you dislike compass points, the names **.top**, **.bottom**, **.left** and **.right** are synonyms for **.n**, **.s**, **.e**, and **.w** respectively; they can even be abbreviated to **.t**, **.b**, **.l** and **.r**).

The names **center**, **top**, **bottom**, **left**, **right**, **north**, **south**, **east**, and **west** can also be used (without the leading dot) in a prefix form marked by **of**; thus, **center of last circle** and **top of 2nd last ellipse** are both valid object references. Finally, the names **left** and **right** can be prefixed with **upper** and **lower** which both have the obvious meaning.

Arc objects also have compass points; they are the compass points of the implied circle.

Non-closed objects (line, arrow, or spline) have compass points too, but the locations of them are completely arbitrary. In particular, different **pic** implementations return different locations.

6.3.12.2.2. Locations Relative to Open Objects

Every open object (line, arrow, arc, or spline) has three named points: **.start**, **.center** (or **.c**), and **.end**. They can also be used without leading dots in the **of** prefix form. The center of an arc is the center of its circle, but the center of a line, path, or spline is halfway between its endpoints.

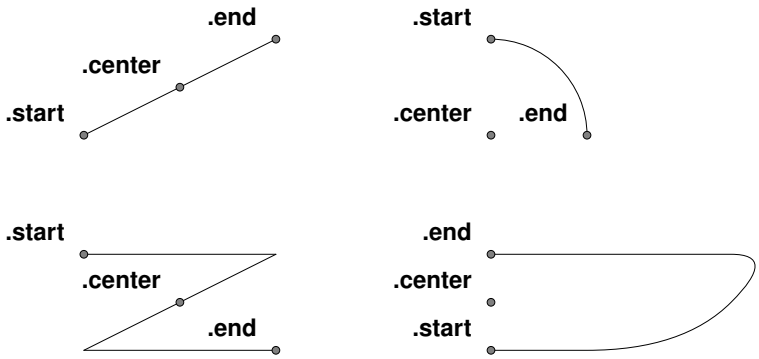


Figure 6-2: Special points on open objects

6.3.12.3. Ways of Composing Positions

Once you have two positions to work with, there are several ways to combine them to specify new positions.

6.3.12.3.1. Vector Sums and Displacements

Positions may be added or subtracted to yield a new position (to be more precise, you can only add a position and an expression pair; the latter must be on the right side of the addition or subtraction sign). The result is the conventional vector sum or difference of coordinates. For example, **last box .ne + (0.1, 0)** is a valid position. This example illustrates a common use, to define a position slightly offset from a named one (say, for captioning purposes).

6.3.12.3.2. Interpolation Between Positions

A position may be interpolated between any two positions. The syntax is '*fraction* **of the way between** *position1* **and** *position2*'. For example, you can say **1/3 of the way between Here and last ellipse .ne**. The fraction may be in numerator/denominator form or may be an ordinary number (values are *not* restricted to [0,1]). As an alternative to this verbose syntax, you can say '*fraction* **<***position1* **,** *position2***>**'; thus, the example could also be written as**1/3 <Here, last ellipse>**.

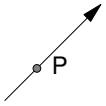


Figure 6-3: **P: 1/3 of the way between last arrow .start and last arrow .end**

This facility can be used, for example, to draw double connections.

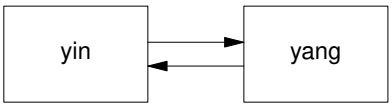


Figure 6-4: Doubled arrows

You can get Figure 6-4 from the following program:

```
.PS
A: box "yin"; move;
B: box "yang";
arrow right at 1/4 <A.e,A.ne>;
arrow left  at 1/4 <B.w,B.sw>;
.PE
```

Note the use of the short form for interpolating points.

6.3.12.3.3. Projections of Points

Given two positions *p* and *q*, the position **(***p***,** *q***)** has the X coordinate of *p* and the Y coordinate of *q*. This can be helpful in placing an object at one of the corners of the virtual box defined by two other objects.

Figure 6-5: Using (*x*, *y*) composition

6.3.12.4. Using Locations

There are four ways to use locations; **at**, **from**, **to**, and **with**. All four are object modifiers; that is, you use them as suffixes to a drawing command.

The **at** modifier says to draw a closed object or arc with its center at the following location, or to draw a line/spline/arrow starting at the following location.

The **to** modifier can be used alone to specify a move destination. The **from** modifier can be used alone in the same way as **at**.

The **from** and **to** modifiers can be used with a **line** or **arc** command to specify start and end points of the object. In conjunction with named locations, this offers a very flexible mechanism for connecting objects. For example, the following program

```
.PS
box "from"
move 0.75;
ellipse "to"
arc cw from 1/3 of the way \
    between last box .n and last box .ne to last ellipse .n;
.PE
```

yields:

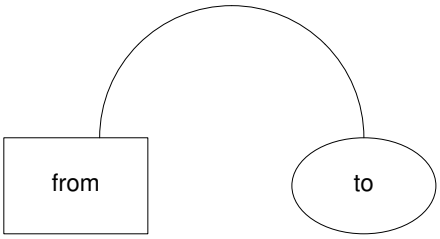
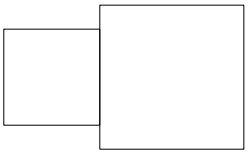
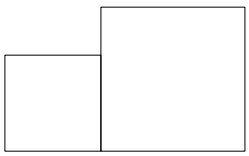


Figure 6-6: A tricky connection specified with English-like syntax

The **with** modifier allows you to identify a named attachment point of an object (or a position within the object) with another point. This is very useful for connecting objects in a natural way. For an example, consider these two programs:



box wid 0.5 ht 0.5; box wid 0.75 ht 0.75



box wid 0.5 ht 0.5;
box wid 0.75 ht 0.75 with .sw at last box .se;

Figure 6-7: Using the **with** modifier for attachments

6.3.12.5. The ‘chop’ Modifier

When drawing lines between circles that don’t intersect them at a compass point, it is useful to be able to shorten a line by the radius of the circle at either or both ends. Consider the following program:

```
.PS
circle "x"
circle "y" at 1st circle - (0.4, 0.6)
circle "z" at 1st circle + (0.4, -0.6)
arrow from 1st circle to 2nd circle chop
arrow from 2nd circle to 3rd circle chop
arrow from 3rd circle to 1st circle chop
.PE
```

It yields the following:

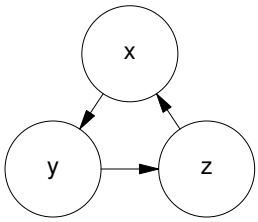


Figure 6-8: The **chop** modifier

Notice that the **chop** attribute moves arrowheads rather than stepping on them. By default, the **chop** modifier shortens both ends of the line by **circlerad**. By suffixing it with a number you can change the amount of chopping.

If you say **line ... chop** *r1* **chop** *r2* with *r1* and *r2* both numbers, you can vary the amount of chopping at both ends. You can use this in combination with trigonometric functions to write code that deals with more complex intersections.

6.3.13. Object Groups

There are two different ways to group objects in **pic**; *brace grouping* and *block composites*.

6.3.13.1. Brace Grouping

The simpler method is simply to group a set of objects within curly bracket or brace characters. On exit from this grouping, the current position and direction are restored to their value when the opening brace was encountered.

6.3.13.2. Block Composites

A block composite object is created a series of commands enclosed by square brackets. The composite can be treated for most purposes like a single closed object, with the size and shape of its bounding box. Here is an example. The program fragment

```
A: [
    circle;
    line up 1 at last circle .n;
    line down 1 at last circle .s;
    line right 1 at last circle .e;
    line left 1 at last circle .w;
    box dashed with .nw at last circle .se + (0.2, -0.2);
    Caption: center of last box;
]
```

yields the block in figure 6-1, which we show both with and without its attachment points. The block's location becomes the value of **A**.

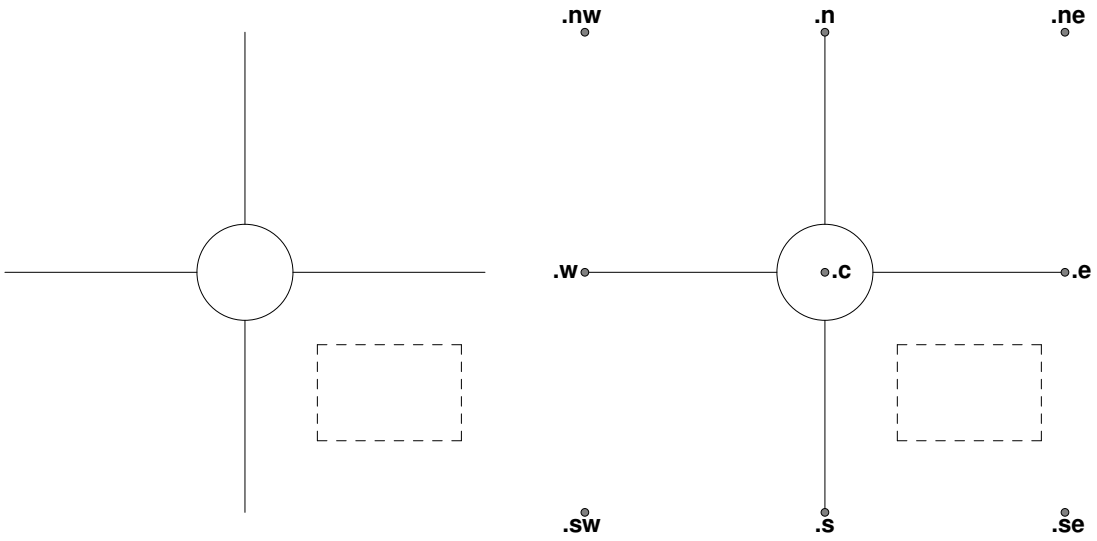


Figure 6-1: A sample composite object

To refer to one of the composite's attachment points, you can say (for example) **A .s**. For purposes of object naming, composites are a class. You could write **last [] .s** as an equivalent reference, usable anywhere a location is needed. This construction is very important for putting together large, multi-part diagrams.

Blocks are also a variable-scoping mechanism, like a *groff* (1) environment. All variable assignments done inside a block are undone at the end of it. To get at values within a block, write a name of the block followed by a dot, followed by the label you want. For example, we could refer the center of the box in the above composite as **last [] .Caption** or **A.Caption**.

This kind of reference to a label can be used in any way any other location can be. For example, if we added **"Hi!" at A.Caption** the result would look like this:

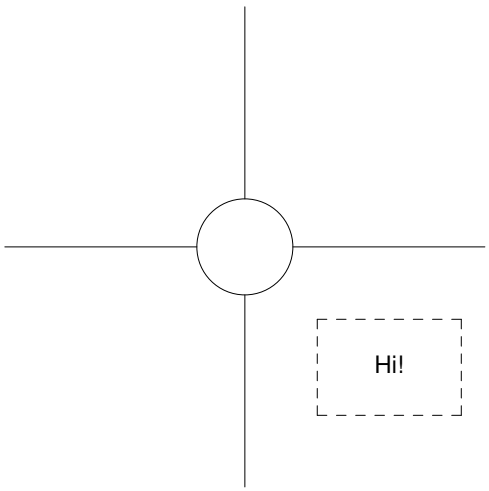
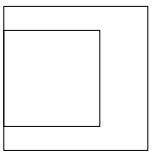


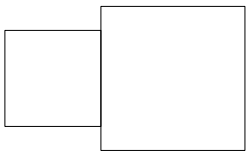
Figure 6-2: Adding a caption using interior labeling

You can also use interior labels in either part of a **with** modifier. This means that the example composite could be placed relative to its caption box by a command containing **with A.Caption at**.

Note that both width and height of the block composite object are always positive:



box wid -0.5 ht 0.5; box wid 0.75 ht 0.75



[box wid -0.5 ht 0.5]; box wid 0.75 ht 0.75

Figure 6-3: Composite block objects always have positive width and height

Blocks may be nested. This means you can use block attachment points to build up complex diagrams hierarchically, from the inside out. Note that **last** and the other sequential naming mechanisms don't look inside blocks, so if you have a program that looks like

```
.PS
P: [box "foo"; ellipse "bar"];
Q: [
    [box "baz"; ellipse "quxx"]
    "random text";
   ]
arrow from 2nd last [];
.PE
```

the arrow in the last line is attached to object **P**, not object **Q**.

In DWB **pic**, only references one level deep into enclosed blocks were permitted. GNU **gpic** removes this restriction.

The combination of block variable scoping, assignability of labels and the macro facility that we'll describe later on can be used to simulate functions with local variables (just wrap the macro body in block braces).

6.3.14. Style Variables

There are a number of global style variables in **pic** that can be used to change its overall behavior. We've mentioned several of them in previous sections. They're all described here. For each variable, the default is given.

Style Variable	Default	What It Does
boxht	0.5	Default height of a box
boxwid	0.75	Default width of a box
lineht	0.5	Default length of vertical line
linewid	0.75	Default length of horizontal line
linethick	-1	Default line thickness
arcrad	0.25	Default radius of an arc
circlerad	0.25	Default radius of a circle
ellipseht	0.5	Default height of an ellipse
ellipsewid	0.75	Default width of an ellipse
moveht	0.5	Default length of vertical move
movewid	0.75	Default length of horizontal move
textht	0	Default height of box enclosing a text object
textwid	0	Default width of box enclosing a text object
arrowht	0.1	Length of arrowhead along shaft
arrowwid	0.05	Width of rear of arrowhead
arrowhead	1	Enable/disable arrowhead filling
dashwid	0.05	Interval for dashed lines
maxpswid	8.5	Maximum width of picture
maxpsht	11	Maximum height of picture
scale	1	Unit scale factor
fillval	0.5	Default fill value

Any of these variables can be set with a simple assignment statement. For example:

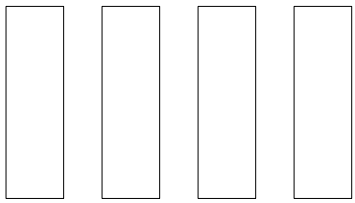


Figure 6-1: **boxht=1; boxwid=0.3; movewid=0.2; box; move; box; move; box; move; box;**

In GNU **pic**, setting the **scale** variable re-scales all size-related state variables so that their values remain equivalent in the new units.

The command **reset** resets all style variables to their defaults. You can give it a list of variable names as arguments (optionally separated by commas), in which case it resets only those.

State variables retain their values across pictures until reset.

6.3.15. Expressions, Variables, and Assignment

A number is a valid expression, of course (all numbers are stored internally as floating-point). Decimal-point notation is acceptable; in GNU **gpic**, scientific notation in C's 'e' format (like `5e-2`) is accepted.

Anywhere a number is expected, the language also accepts a variable. Variables may be the built-in style variable described in the last section, or new variables created by assignment.

DWB **pic** supports only the ordinary assignment via **=**, which defines the variable (on the left side of the equal sign) in the current block if it is not already defined there, and then changes the value (on the right side) in the current block. The variable is not visible outside of the block. This is similar to the C programming language where a variable within a block shadows a variable with the same name outside of the block.

GNU **gpic** supports an alternate form of assignment using **:=**. The variable must already be defined, and the value is assigned to that variable without creating a variable local to the current block. For example, this

```
x=5
y=5
[
  x:=3
  y=3
]
print x " " y
```

prints **3 5**.

You can use the height, width, radius, and x and y coordinates of any object or corner in expressions. If**A** is an object label or name, all the following are valid:

```
A.x                 # x coordinate of the center of A
A.ne.y              # y coordinate of the northeast corner of A
A.wid               # the width of A
A.ht                # and its height
2nd last circle.rad  # the radius of the 2nd last circle
```

Note the second expression, showing how to extract a corner coordinate.

Basic arithmetic resembling those of C operators are available; **+**, *****, **-**, /, and**%**. So isˆ f or exponentiation. Grouping is permitted in the usual way using parentheses. GNU**gpic** allo ws logical operators to appear in expressions; **!** (logical negation, not factorial), **&&**, ||, **==**, **!=**, **>=**, **<=**, **<**, **>**.

Various built-in functions are supported: **sin(***x***)**, **cos(***x***)**, **log(***x***)**, **exp(***x***)**, **sqrt(***x***)**, **max(***x***,***y***)**, **atan2(***x***,***y***)**, **min(***x***,***y***)**, **int(***x***)**, **rand()**, and **srand()**. Both**e xp** and **log** are base 10; **int** does integer truncation; **rand()** returns a random number in [0-1), and **srand()** sets the seed for a new sequence of pseudo-random numbers to be returned by **rand()** (**srand()** is a GNU extension).

GNU **gpic** also documents a one-argument form or rand, **rand(***x***)**, which returns a random number between 1 and *x*, but this is deprecated and may be removed in a future version.

The function **sprintf()** behaves like a C *sprintf*(3) function that only takes %%, %e, %E, %f, %g, and %G conversion specifications.

6.3.16. Macros

You can define macros in **pic**, with up to 32 arguments (up to 16 on EBCDIC platforms). This is useful for diagrams with repetitive parts. In conjunction with the scope rules for block composites, it effectively gives you the ability to write functions.

The syntax is

define *name* **{** *replacement text* **}**

This defines *name* as a macro to be replaced by the replacement text (not including the braces). The macro may be called as

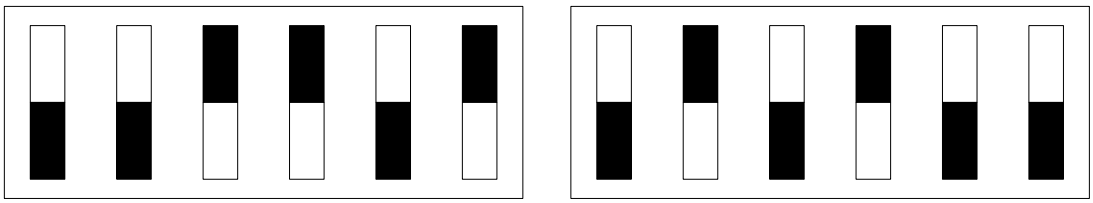
name(*arg1, arg2, . . . argn*)

The arguments (if any) are substituted for tokens **$1**, **$2** . . . **$n** appearing in the replacement text.

As an example of macro use, consider this:

```
.PS
# Plot a single jumper in a box, $1 is the on-off state.
define jumper { [
    shrinkfactor = 0.8;
    Outer: box invis wid 0.45 ht 1;

    # Count on end ] to reset these
    boxwid = Outer.wid * shrinkfactor / 2;
    boxht  = Outer.ht  * shrinkfactor / 2;

    box fill (!$1) with .s at center of Outer;
    box fill ($1)  with .n at center of Outer;
] }

# Plot a block of six jumpers.
define jumperblock {
    jumper($1);
    jumper($2);
    jumper($3);
    jumper($4);
    jumper($5);
    jumper($6);

    jwidth  = last [].Outer.wid;
    jheight = last [].Outer.ht;

    box with .nw at 6th last [].nw wid 6*jwidth ht jheight;

    # Use {} to avoid changing position from last box draw.
    # This is necessary so move in any direction works as expected
    {"Jumpers in state $1$2$3$4$5$6" at last box .s + (0,-0.2);}
}

# Sample macro invocations.
jumperblock(1,1,0,0,1,0);
move;
jumperblock(1,0,1,0,1,1);
.PE
```

It yields the following:



Jumpers in state 110010 Jumpers in state 101011

Figure 6-1: Sample use of a macro

This macro example illustrates how you can combine [], brace grouping, and variable assignment to write true functions.

One detail the example above does not illustrate is the fact that macro argument parsing is not token-oriented. If you call **jumper(1)**, the value of $1 is **" 1 "**. You could even call **jumper(big string)** to give $1 the value **"big string"**.

If you want to pass in a coordinate pair, you can avoid getting tripped up by the comma by wrapping the pair in parentheses.

Macros persist through pictures. To undefine a macro, say **undef** *name*; for example,

```
undef jumper
undef jumperblock
```

would undefine the two macros in the jumper block example.

6.3.17. Import/Export Commands

Commands that import or export data between **pic** and its environment are described here.

6.3.17.1. File and Table Insertion

The statement

`copy` *filename*

inserts the contents of *filename* in the **pic** input stream. Any **.PS**/**.PE** pair in the file is ignored. You can use this to include pre-generated images.

A variant of this statement replicates the **copy thru** feature of *grap*(1). The call

`copy` *filename* `thru` *macro*

calls *macro* (which may be either a name or replacement text) on the arguments obtained by breaking each line of the file into blank-separated fields. The macro may have up to 9 arguments. The replacement text may be delimited by braces or by a pair of instances of any character not appearing in the rest of the text.

If you write

`copy thru` *macro*

omitting the filename, lines to be parsed are taken from the input source up to the next **.PE**.

In either of the last two **copy** commands, GNU **gpic** permits a trailing '**until** *word*' clause to be added which terminates the copy when the first word matches the argument (the default behavior is therefore equivalent to **until .PE**).

Accordingly, the command

```
.PS
copy thru % circle at ($1,$2) % until "END"
1 2
3 4
5 6
END
box
.PE
```

is equivalent to

```
.PS
circle at (1,2)
circle at (3,4)
circle at (5,6)
box
.PE
```

6.3.17.2. Debug Messages

The command **print** accepts any number of arguments, concatenates their output forms, and writes the result to standard error. Each argument must be an expression, a position, or a text string.

6.3.17.3. Escape to Post-Processor

If you write

command *arg*...

pic concatenates the arguments and pass them through as a line to troff or TEX. Each*arg* m ust be an expression, a position, or text. This has a similar effect to a line beginning with **.** or \, but allows the values of variables to be passed through.

For example,

```
.PS
x = 14
command ".ds string x is " x "."
.PE
\*[string]
```

prints

```
x is 14.
```

6.3.17.4. Executing Shell Commands

The command

sh { *anything*... }

macro-expands the text in braces, then executes it as a shell command. This could be used to generate images or data tables for later inclusion. The delimiters shown as {} here may also be two copies of any one character not present in the shell command text. In either case, the body may contain balanced {} pairs. Strings in the body may contain balanced or unbalanced braces in any case.

6.3.18. Control-flow constructs

The **pic** language provides conditionals and looping. For example,

```
pi = atan2(0,-1);
for i = 0 to 2 * pi by 0.1 do {
    "-" at (i/2, 0);
    "." at (i/2, sin(i)/2);
    ":" at (i/2, cos(i)/2);
}
```

which yields this:

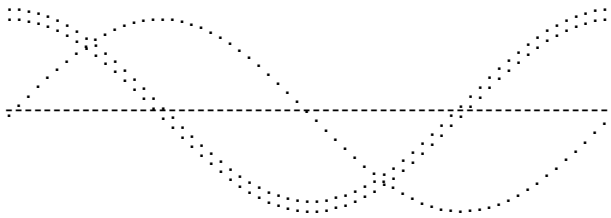


Figure 6-1: Plotting with a **for** loop

The syntax of the **for** statement is:

for *variable* **=** *expr1* **to** *expr2* [**by** [*]*expr3*] **do** *X body X*

The semantics are as follows: Set*v ariable* to *expr1*. While the value of *variable* is less than or equal to *expr2*, do *body* and increment *variable* by *expr3*; if **by** is not given, increment *variable* by 1. If*e xpr3* is prefixed by * then *variable* is multiplied instead by *expr3*. The value of *expr3* can be negative for the additive case; *variable* is then tested whether it is greater than or equal to *expr2*. For the multiplicative case, *expr3* must be greater than zero. If the constraints aren't met, the loop isn't executed. *X* can be any character not occurring in *body*; or the two *X*s may be paired braces (as in the **sh** command).

The syntax of the **if** statement is as follows:

if *expr* **then** *X if-true X* [**else** *Y if-false Y*]

Its semantics are as follows: Evaluate *expr*; if it is non-zero then do *if-true*, otherwise do *if-false*. *X* can be any character not occurring in *if-true*. *Y* can be any character not occurring in *if-false*.

Eithe or both of the*X* or *Y* pairs ma y instead be balanced pairs of braces ({ and }) as in the **sh** command. In either case, the *if-true* may contain balanced pairs of braces. None of these delimiters are seen inside strings.

All the usual relational operators my be used in conditional expressions; **!** (logical negation, not factorial), **&&**, ||, **==**, **!=**, **>=**, **<=**, **<**, **>**.

String comparison is also supported using**==** and **!=**. String comparisons ma y need to be parenthesized to avoid syntactic ambiguities.

6.3.19. Interface To [gt]roff

The output of **pic** is **[gt]roff** drawing commands. The GNU*gpic*(1) command w arns that it relies on drawing extensions present in *groff*(1) that are not present in *troff*(1).

6.3.19.1. Scaling Arguments

The DWB *pic*(1) program accepts one or two arguments to **.PS**, which is interpreted as a width and height in inches to which the results of *pic*(1) should be scaled (width and height scale independently). If there is only one argument, it is interpreted as a width to scale the picture to, and height is scaled by the same proportion.

GNU **gpic** is less general; it accepts a single width to scale to, or a zero width and a maximum height to scale to. With two non-zero arguments, it scales to the maximum height.

6.3.19.2. How Scaling is Handled

When **pic** processes a picture description on input, it passes **.PS** and **.PE** through to the postprocessor. The**.PS** gets decorated with two numeric arguments which are the X and Y dimensions of the picture in inches. The post-processor can use these to reserve space for the picture and center it.

The GNU incarnation of the **ms** macro package, for example, includes the following definitions:

```
.de PS
.br
.sp \\n[DD]u
.ie \\n[.$]<2 .@error bad arguments to PS (not preprocessed with pic?)
.el \{\
.    ds@need (u;\\$1)+1v
.    in +(u;\\n[.l]-\\n[.i]-\\$2/2>?0)
.\}
..
.de PE
.par@reset
.sp \\n[DD]u+.5m
..
```

Equivalent definition is supplied by GNU *pic*(1) if you use the –mpic option; this should make it usable with macro pages other than *ms*(1).

If **.PF** is used instead of **.PE**, the **troff** position is restored to what it was at the picture start (Kernighan notes that the F stands for “flyback”).

The invocation

.PS <*file*

causes the contents of *file* to replace the **.PS** line. This feature is deprecated; use ‘**copy** *file*’ instead).

6.3.19.3. PIC and [gt]roff commands

By default, input lines that begin with a period are passed to the postprocessor, embedded at the corresponding point in the output. Messing with horizontal or vertical spacing is an obvious recipe for bugs, but point size and font changes are usually safe.

Point sizes and font changes are also safe within text strings, as long as they are undone before the end of string.

The state of **[gt]roff**’s fill mode is preserved across pictures.

6.3.19.4. PIC and EQN

The Kernighan paper notes that there is a subtle problem with complicated equations inside **pic** pictures; they come out wrong if *eqn*(1) has to leave extra vertical space for the equation. If your equation involves more than subscripts and superscripts, you must add to the beginning of each equation the extra information **space 0**. He gives the following example:

```
arrow
box "$space 0 {H( omega )} over {1 - H( omega )}$"
arrow
```

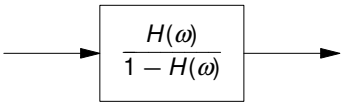


Figure 6-1: Equations within pictures

6.3.19.5. Absolute Positioning of Pictures

A **pic** picture is positioned vertically by troff at the current position. The topmost position possible on a page is not the paper edge but a position which is one baseline lower so that the first row of glyphs is visible. To make a picture really start at the paper edge you have to make the baseline-to-baseline distance zero, this is, you must set the vertical spacing to 0 (using **.vs**) before starting the picture.

6.3.20. Interface to TeX

TEX mode is enabled by the **–t** option. In TEX mode, pic defines a vbox called **graph** for each picture; the name can be changed with the pseudo-variable **figname** (which is actually a specially parsed command). You must yourself print that vbox using, for example, the command

```
\centerline{\box\graph}
```

Actually, since the vbox has a height of zero (it is defined with \vtop) this produces slightly more vertical space above the picture than below it;

```
\centerline{\raise 1em\box\graph}
```

would avoid this.

To make the vbox having a positive height and a depth of zero (as used e.g. by LATEX's `graphics.sty`), define the following macro in your document:

```
\def\gpicbox#1{%
  \vbox{\unvbox\csname #1\endcsname\kern 0pt}}
```

Now you can simply say **gpicbox{graph}** instead of \box\graph.

You must use a TEX driver that supports the **tpic** specials, version 2.

Lines beginning with \ are passed through transparently; a **%** is added to the end of the line to avoid unwanted spaces. You can safely use this feature to change fonts or to change the value of **baselineskip**. Anything else may well produce undesirable results; use at your own risk. Lines beginning with a period are not given any special treatment.

The TEX mode of *pic*(1) does *not* translate **troff** font and size changes included in text strings!

Here an example how to use **figname**.

```
.PS
figname = foo;
...
.PE

.PS
figname = bar;
...
.PE

\centerline{\box\foo \hss \box\bar}
```

Use this feature sparsingly and only if really needed: A different name means a new box register in TEX, and the maximum number of box registers is only 256. Also be careful not to use a predefined TEX or LATEX macro name as an argument to **figname** since this inevitably causes an error.

6.3.21. Obsolete Commands

GNU *gpic*(1) has a command

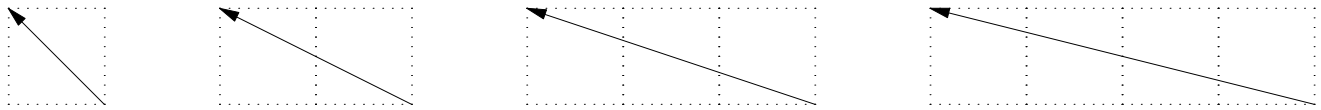
plot *expr* [**"***text***"**]

This is a text object which is constructed by using *text* as a format string for sprintf with an argument of *expr*. If*te xt* is omitted a format string of **"%g"** is used. Attributes can be specified in the same way as for a normal text object. Be very careful that you specify an appropriate format string; **pic** does only very limited checking of the string. This is deprecated in favour of **sprintf**.

6.3.22. Some Larger Examples

Here are a few larger examples, with complete source code. One of our earlier examples is generated in an instructive way using a for loop:

```
.PS
# Draw a demonstration up left arrow with grid box overlay
define gridarrow
{
    move right 0.1
    [
        {arrow up left $1;}
        box wid 0.5 ht 0.5 dotted with .nw at last arrow .end;
        for i = 2 to ($1 / 0.5) do
        {
            box wid 0.5 ht 0.5 dotted with .sw at last box .se;
        }
        move down from last arrow .center;
        [
            sprintf("\fBarrow up left %g\fP", $1)
        ]
    ]
    move right 0.1 from last [] .e;
}
gridarrow(0.5);
gridarrow(1);
gridarrow(1.5);
gridarrow(2);
undef gridarrow
.PE
```



arrow up left 0.5 **arrow up left 1** **arrow up left 1.5** **arrow up left 2**

Figure 6-1: Diagonal arrows (dotted boxes show the implied 0.5-inch grid)

Here's an example concocted to demonstrate layout of a large, multiple-part pattern:

```
.PS
define filter {box ht 0.25 rad 0.125}
lineht = 0.25;
Top: [
    right;
    box "\fBms\fR" "sources";
    move;
    box "\fBHTML\fR" "sources";
    move;
    box "\fBlinuxdoc-sgml\fP" "sources" wid 1.5;
    move;
    box "\fBTexinfo\fP" "sources";

    line down from 1st box .s lineht;
    A: line down;
    line down from 2nd box .s; filter "\fBhtml2ms\fP";
    B: line down;
    line down from 3rd box .s; filter "\fBformat\fP";
    C: line down;
    line down from 4th box .s; filter "\fBtexi2roff\fP";
    D: line down;
]
move down 1 from last [] .s;
Anchor: box wid 1 ht 0.75 "\fBms\fR" "intermediate" "form";
arrow from Top.A.end to Anchor.nw;
arrow from Top.B.end to 1/3 of the way between Anchor.nw and Anchor.ne;
arrow from Top.C.end to 2/3 of the way between Anchor.nw and Anchor.ne;
arrow from Top.D.end to Anchor.ne
{
    # PostScript column
    move to Anchor .sw;
    line down left then down ->;
    filter "\fBpic\fP";
    arrow;
    filter "\fBeqn\fP";
    arrow;
    filter "\fBtbl\fP";
    arrow;
    filter "\fBgroff\fP";
    arrow;
    box "PostScript";

    # HTML column
    move to Anchor .se;
    line down right then down ->;
    A: filter dotted "\fBpic2img\fP";
    arrow;
    B: filter dotted "\fBeqn2html\fP";
    arrow;
    C: filter dotted "\fBtbl2html\fP";
    arrow;
    filter "\fBms2html\fP";
    arrow;
    box "HTML";

    # Nonexistence caption
    box dashed wid 1 at B + (2,0) "These tools" "don't yet exist";
    line chop 0 chop 0.1 dashed from last box .nw to A.e ->;
    line chop 0 chop 0.1 dashed from last box .w  to B.e ->;
    line chop 0 chop 0.1 dashed from last box .sw to C.e ->;
}
.PE
```

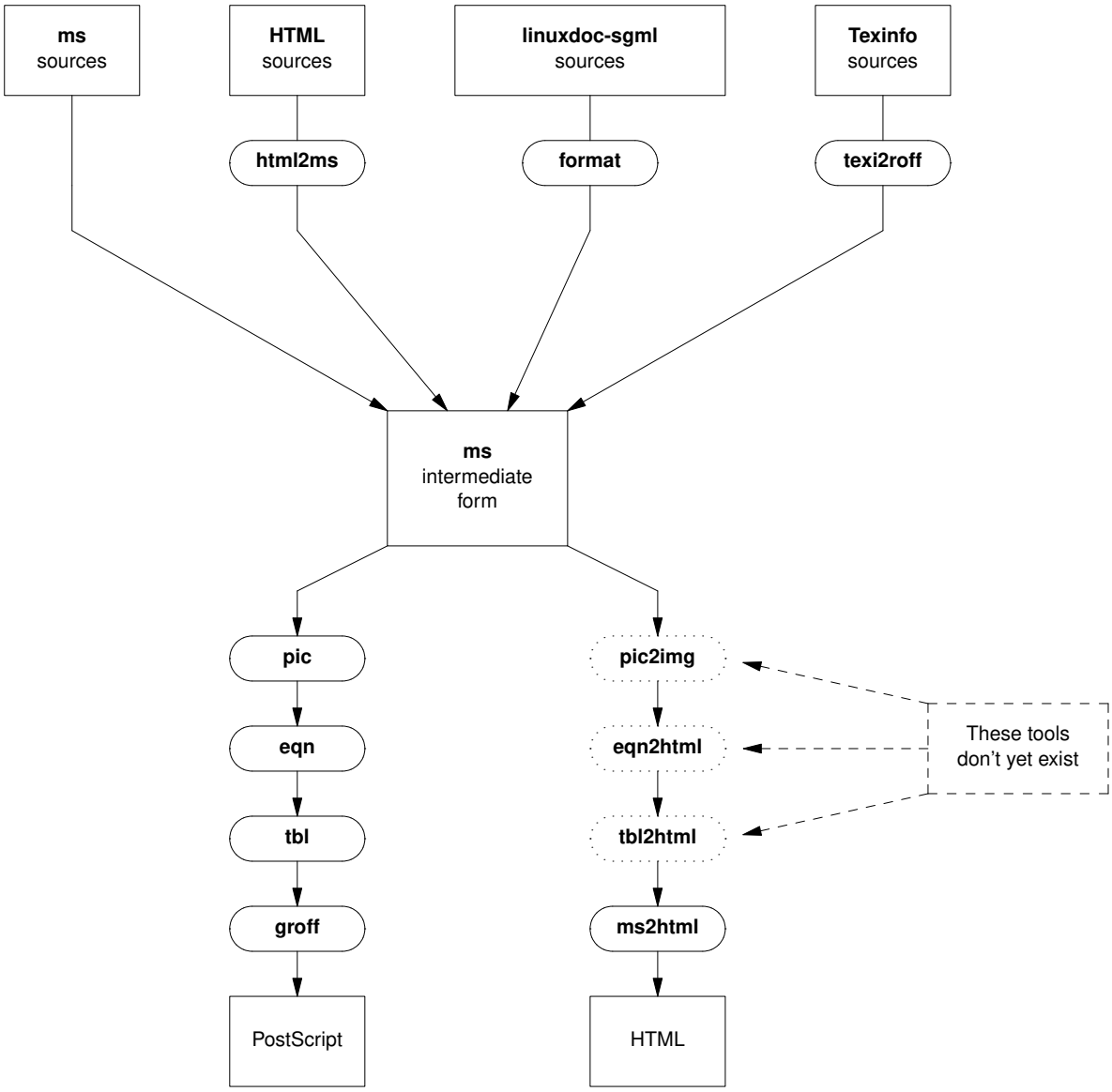


Figure 6-2: Hypothetical production flow for dual-mode publishing

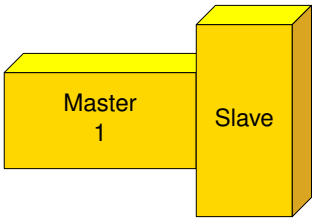


Figure 6-3: Three-dimensional Boxes

Here the source code for figure 6-3:

```
.PS
# a three-dimensional block
#
# tblock(<width>, <height>, <text>)

define tblock { [
  box ht $2 wid $1 \
      color "gold" outlined "black" \
      xslanted 0 yslanted 0 \
      $3;
  box ht .1 wid $1 \
      color "yellow" outlined "black" \
      xslanted .1 yslanted 0 \
      with .sw at last box .nw;
  box ht $2 wid .1 \
      color "goldenrod" outlined "black" \
      xslanted 0 yslanted .1 \
      with .nw at 2nd last box .ne;
] }

tblock(1, .5, "Master" "1");
move -.1
tblock(.5, 1, "Slave");
.PE
```

6.3.23. PIC Reference

This is an annotated grammar of **pic**.

6.3.23.1. Lexical Items

In general, **pic** is a free-format, token-oriented language that ignores whitespace outside strings. But certain lines and contructs are specially interpreted at the lexical level:

A comment begins with **#** and continues to **n** (comments may also follow text in a line). A line beginning with a period or backslash may be interpreted as text to be passed through to the post-processor, depending on command-line options. An end-of-line backslash is interpreted as a request to continue the line; the backslash and following newline are ignored.

Here are the grammar terminals:

INT A positive integer.

NUMBER
: A floating point numeric constant. May contain a decimal point or be expressed in scientific notation in the style of *printf*(3)'s %e escape. A trailing 'i' or 'I' (indicating the unit 'inch') is ignored.

TEXT A string enclosed in double quotes. A double quote within TEXT must be preceded by a backslash. Instead of TEXT you can use

```
sprintf ( TEXT [, <expr> ...] )
```

except after the 'until' and 'last' keywords, and after all ordinal keywords ('th' and friends).

VARIABLE
: A string starting with a character from the set [a-z], optionally followed by one or more characters of the set [a-zA-Z0-9_]. (Values of variables are preserved across pictures.)

LABEL A string starting with a character from the set [A-Z], optionally followed by one or more characters of the set [a-zA-Z0-9_].

COMMAND-LINE
: A line starting with a command character ('.' in groff mode, '\' in TEX mode).

BALANCED-TEXT
: A string either enclosed by '{' and '}' or with *X* and *X*, where *X* doesn't occur in the string.

BALANCED-BODY
: Delimiters as in BALANCED-TEXT; the body is interpreted as '⟨**command**⟩**...**'.

FILENAME
: The name of a file. This has the same semantics as TEXT.

MACRONAME
: Either VARIABLE or LABEL.

6.3.23.2. Semi-Formal Grammar

Tokens not enclosed in ⟨ ⟩ are literals, except:

1. **n** is a newline.
2. Three dots is a suffix meaning 'replace with 0 or more repetitions of the preceding element(s).
3. An enclosure in square brackets has its usual meaning of 'this clause is optional'.
4. Square-bracket-enclosed portions within tokens are optional. Thus, 'h[eigh]t' matches either 'height' or 'ht'.

If one of these special tokens has to be referred to literally, it is surrounded with single quotes.

The top-level **pic** object is a picture.

```
<picture> ::=
  .PS [NUMBER [NUMBER]]\n
  <statement> ...
  .PE \n
```

The arguments, if present, represent the width and height of the picture, causing **pic** to attempt to scale it to the given dimensions in inches. In no case, however, the X and Y dimensions of the picture exceed the values of the style variables **maxpswid** and **maxpsheight** (which default to the normal 8.5i by 11i page size).

If the ending '.PE' is replaced by '.PF', the page vertical position is restored to its value at the time '.PS' was encountered. Another alternate form of invocation is '.PS <FILENAME', which replaces the '.PS' line with a file to be interpreted by **pic** (but this feature is deprecated).

The '.PS', '.PE', and '.PF' macros to perform centering and scaling are normally supplied by the post-processor.

In the following, either '|' or a new line starts an alternative.

```
<statement> ::=
  <command> ;
  <command> \n


<command> ::=
  <primitive> [<attribute>]
  LABEL : [;] <command>
  LABEL : [;] <command> [<position>]
  { <command> ... }
  VARIABLE [:] = <any-expr>
  figname = MACRONAME
  up | down | left | right
  COMMAND-LINE
  command <print-arg> ...
  print <print-arg> ...
  sh BALANCED-TEXT
  copy FILENAME
  copy [FILENAME] thru MACRONAME [until TEXT]
  copy [FILENAME] thru BALANCED-BODY [until TEXT]
  for VARIABLE = <expr> to <expr> [by [*] <expr>] do BALANCED-BODY
  if <any-expr> then BALANCED-BODY [else BALANCED-BODY]
  reset [VARIABLE [[,] VARIABLE ...]]


<print-arg> ::=
  TEXT
  <expr>
  <position>
```

The current position and direction are saved on entry to a '{ . . . }' construction and restored on exit from it.

Note that in 'if' constructions, newlines can only occur in BALANCED-BODY. This means that

```
if
{ ... }
else
{ ... }
```

fails. You have to use the braces on the same line as the keywords:

```
if {
...
} else {
...
}
```

This restriction doesn't hold for the body after the 'do' in a 'for' construction.

At the beginning of each picture, 'figname' is reset to the vbox name 'graph'; this command has only a meaning in TEX mode. While the grammar rules allow digits and the underscore in the value of 'figname', TEX normally accepts uppercase and lowercase letters only as box names (you have to use '\csname' if you really need to circumvent this limitation).

```
<any-expr> ::=
  <expr>
  <text-expr>
  <any-expr> <logical-op> <any-expr>
  ! <any-expr>

<logical-op> ::=
  == | != | && | '||'

<text-expr> ::=
  TEXT == TEXT
  TEXT != TEXT
```

Logical operators are handled specially by **pic** since they can deal with text strings also. **pic** uses *strcmp*(3) to test for equality of strings; an empty string is considered as 'false' for '&&' and '||'.

```
<primitive> ::=
  box                   # closed object — rectangle
  circle                # closed object — circle
  ellipse               # closed object — ellipse
  arc                   # open object — quarter-circle
  line                  # open object — line
  arrow                 # open object — line with arrowhead
  spline                # open object — spline curve
  move
  TEXT TEXT ...         # text within invisible box
  plot <expr> TEXT      # formatted text
  '[' <command> ... ']'
```

Drawn objects within '[. . .]' are treated as a single composite object with a rectangular shape (that of the bounding box of all the elements). Variable and label assignments within a block are local to the block. Current direction of motion is restored to the value at start of block upon exit. Position is *not* restored (unlike '{ }'); instead, the current position becomes the exit position for the current direction on the block's bounding box.

```
<attribute> ::=
  h[eigh]t <expr>          # set height of closed figure
  wid[th] <expr>           # set width of closed figure
  rad[ius] <expr>          # set radius of circle/arc
  diam[eter] <expr>        # set diameter of circle/arc
  up [<expr>]              # move up
  down [<expr>]            # move down
  left [<expr>]            # move left
  right [<expr>]           # move right
  from <position>          # set from position of open figure
  to <position>            # set to position of open figure
  at <position>            # set center of open figure
  with <path>              # fix corner/named point at specified location
  with <position>          # fix position of object at specified location
  by <expr-pair>           # set object's attachment point
  then                     # sequential segment composition
  dotted [<expr>]          # set dotted line style
  dashed [<expr>]          # set dashed line style
  thick[ness] <expr>       # set thickness of lines
  chop [<expr>]            # chop end(s) of segment
  '->' | '<-' | '<->'      # decorate with arrows
  invis[ible]              # make primitive invisible
  solid                    # make closed figure solid
  fill[ed] [<expr>]        # set fill density for figure
  xscaled <expr>           # slant box into x direction
  yscaled <expr>           # slant box into y direction
  colo[u]r[ed] TEXT        # set fill and outline color for figure
  outline[d] TEXT          # set outline color for figure
  shaded TEXT              # set fill color for figure
  same                     # copy size of previous object
  cw | ccw                 # set orientation of curves
  ljust | rjust            # adjust text horizontally
  above | below            # adjust text vertically
  aligned                  # align parallel to object
  TEXT TEXT ...            # text within object
  <expr>                   # motion in the current direction
```

Missing attributes are supplied from defaults; inappropriate ones are silently ignored. For lines, splines, and arcs, height and width refer to arrowhead size.

The 'at' primitive sets the center of the current object. The 'with' attribute fixes the specified feature of the given object to a specified location. (Note that 'with' is incorrectly described in the Kernighan paper.)

The 'by' primitive is not documented in the tutorial portion of the Kernighan paper, and should probably be considered unreliable.

The primitive 'arrow' is a synonym for 'line ->'.

Text is normally an attribute of some object, in which case successive strings are vertically stacked and centered on the object's center by default. Standalone text is treated as though placed in an invisible box.

A text item consists of a string or sprintf-expression, optionally followed by positioning information. Text (or strings specified with 'sprintf') may contain font changes, size changes, and local motions, provided those changes are undone before the end of the current item. Text may also contain \-escapes denoting special characters. The base font and specific set of escapes supported is implementation dependent, but supported escapes always include the following:

\fR, \f1
: Set Roman style (the default)

\fI, \f2 Set Italic style

\fB, \f3
: Set Bold style

\fP
: Revert to previous style; only works one level deep, does not stack.

Color names are dependent on the pic implementation, but in all modern versions color names recognized by the X window system are supported.

A position is an (x,y) coordinate pair. There are lots of different ways to specify positions:

```
<position> ::=
  <position-not-place>
  <place>
  ( <position> )

<position-not-place> ::=
  <expr-pair>
  <position> + <expr-pair>
  <position> - <expr-pair>
  ( <position> , <position> )
  <expr> [of the way] between <position> and <position>
  <expr> '<' <position> , <position> '>'

<expr-pair> ::=
  <expr> , <expr>
  ( expr-pair )

<place> ::=
  <label>
  <label> <corner>
  <corner> [of] <label>
  Here

<label> ::=
  LABEL [. LABEL ...]
  <nth-primitive>

<corner> ::=
  .n | .e | .w | .s
  .ne | .se | .nw | .sw
  .c[enter] | .start | .end
  .t[op] | .b[ot[tom]] | .l[eft] | .r[ight]
  left | right | <top-of> | <bottom-of>
  <north-of> | <south-of> | <east-of> | <west-of>
  <center-of> | <start-of> | <end-of>
  upper left | lower left | upper right | lower right

<xxx-of> ::=
  xxx                    # followed by 'of'

<nth-primitive> ::=
  <ordinal> <object-type>
  [<ordinal>] last <object-type>

<ordinal> ::=
  INT th
  INT st | INT nd | INT rd
  ` <any-expr> 'th
```

```
<object-type> ::=
  box
  circle
  ellipse
  arc
  line
  arrow
  spline
  '[]'
  TEXT
```

As Kernighan notes, “since barbarisms like **1th** and **3th** are barbaric, synonyms like **1st** and **3rd** are accepted as well.” Objects of a given type are numbered from 1 upwards in order of declaration; the **last** modifier counts backwards.

The “'th” form (which allows you to select a previous object with an expression, as opposed to a numeric literal) is not documented in DWB's *pic*(1).

The ⟨*xxx*-of⟩ rule is special: The lexical parser checks whether *xxx* is followed by the token ‘of’ without eliminating it so that the grammar parser can still see ‘of’. Valid examples of specifying a place with corner and label are thus

```
A .n
.n of A
.n A
north of A
```

while

```
north A
A north
```

both cause a syntax error. (DWB **pic** also allows the weird form ‘A north of’.)

Here the special rules for the ‘with’ keyword using a path:

```
<path> ::=
  <relative-path>
  ( <relative-path> , <relative-path> )

<relative-path> ::=
  <corner>
  . LABEL [. LABEL ...] [<corner>]
```

The following style variables control output:

Style Variable	Default	What It Does
boxht	0.5	Default height of a box
boxwid	0.75	Default width of a box
lineht	0.5	Default length of vertical line
linewid	0.75	Default length of horizontal line
arcrad	0.25	Default radius of an arc
circlerad	0.25	Default radius of a circle
ellipseht	0.5	Default height of an ellipse
ellipsewid	0.75	Default width of an ellipse
moveht	0.5	Default length of vertical move
movewid	0.75	Default length of horizontal move
textht	0	Default height of box enclosing a text object
textwid	0	Default width of box enclosing a text object
arrowht	0.1	Length of arrowhead along shaft
arrowwid	0.05	Width of rear of arrowhead
arrowhead	1	Enable/disable arrowhead filling
dashwid	0.05	Interval for dashed lines
maxpswid	8.5	Maximum width of picture
maxpsht	11	Maximum height of picture
scale	1	Unit scale factor
fillval	0.5	Default fill value

Any of these can be set by assignment, or reset using the **reset** statement. Style variables assigned within '[ ]' blocks are restored to their beginning-of-block value on exit; top-level assignments persist across pictures. Dimensions are divided by **scale** on output.

All **pic** expressions are evaluated in floating point; units are always inches (a trailing 'i' or 'I' is ignored). Expressions have the following simple grammar, with semantics very similar to C expressions:

```
<expr> ::=
  VARIABLE
  NUMBER
  <place> <place-attribute>
  <expr> <op> <expr>
  - <expr>
  ( <any-expr> )
  ! <expr>
  <func1> ( <any-expr> )
  <func2> ( <any-expr> , <any-expr> )
  rand ( )

<place-attribute>
 .x | .y | .h[eigh]t | .wid[th] | .rad

<op> ::=
  + | - | * | / | % | ^ | '<' | '>' | '<=' | '>='

<func1> ::=
  sin | cos | log | exp | sqrt | int | rand | srand

<func2> ::=
  atan2 | max | min
```

Both **exp** and **log** are base 10; **int** does integer truncation; and **rand()** returns a random number in [0-1).

There are **define** and **undef** statements which are not part of the grammar (they behave as pre-processor macros to the language). These may be used to define pseudo-functions.

define *name* **{** *replacement-text* **}**

This defines *name* as a macro to be replaced by the replacement text (not including the braces). The macro may be called as

*name***(***arg1, arg2, . . ., argn***)**

The arguments (if any) are substituted for tokens $1, $2 . . . $n appearing in the replacement text. To undefine a macro, say **undef** *name*, specifying the name to be undefined.

6.3.24. History and Acknowledgements

Original **pic** was written to go with Joseph Ossanna's original *troff*(1) by Brian Kernighan, and later re-written by Kernighan with substantial enhancements (apparently as part of the evolution of *troff*(1) into *ditroff*(1) to generate device-independent output).

The language had been inspired by some earlier graphics languages including **ideal** and **grap**. Kernighan credits Chris van Wyk (the designer of **ideal**) with many of the ideas that went into **pic**.

The **pic** language was originally described by Brian Kernighan in Bell Labs Computing Science Technical Report #116 (you can obtain a PostScript copy of the revised version, [1], by sending a mail message to *netlib@research.att.com* with a body of 'send 116 from research/cstr'). There have been two revisions, in 1984 and 1991.

The document you are reading effectively subsumes Kernighan's description; it was written to fill in lacunæ in the exposition and integrate in descriptions of the GNU *gpic*(1) and *pic2plot*(1) features.

The GNU **gpic** implementation was written by James Clark ⟨*jjc@jclark.com*⟩.

The GNU **pic2plot** implementation is based on James Clark's parser code and maintained by Robert Maier, principal author of **plotutils**.

6.3.25. Bibliography

1. Kernighan, B. W. **PIC — A Graphics Language for Typesetting (Revised User Manual)**. Bell Labs Computing Science Technical Report #116, December 1991.

2. Van Wyk, C. J. **A high-level language for specifying pictures**. *ACM Transactions On Graphics* 1,2 (1982) 163-182.

6.4. ggrn

6.4.1. Invoking ggrn

Name

grn – groff preprocessor for gremlin files

Synopsis

grn [**–C**] [**–T***dev*] [**–M***dir*] [**–F***dir*] [*file* . . .]
grn –?
grn ––help
grn –v
grn ––version

Description

grn is a preprocessor for including *gremlin* pictures in *groff* input. *grn* writes to standard output, processing only input lines between two that start with **.GS** and **.GE**. Those lines must contain *grn* commands (see below). These commands request a *gremlin* file, and the picture in that file is converted and placed in the *troff* input stream. The**.GS** request ma y be followed by a C, L, or R to center, left, or right justify the whole *gremlin* picture (default justification is center). If no *file* is mentioned, the standard input is read. At the end of the picture, the position on the page is the bottom of the *gremlin* picture. If the*g rn* entry is ended with **.GF** instead of **.GE**, the position is left at the top of the picture. Please note that currently only the –me macro package has support for **.GS**, **.GE**, and **.GF**.

Options

–? and **––help** display a usage message, while **–v** and **––version** show version information; all exit afterward.

–T*dev* Prepare output for printer *dev*. The default device is **ps**. See**gr off**(1) for acceptable devices.

–M*dir* Prepend *dir* to the default search path for *gremlin* files. The default path is (in that order) the current directory, the home directory, */usr/local/lib/groff/site–tmac*, */usr/local/share/groff/site–tmac*, and */usr/local/share/groff/1.22.4/tmac*.

–F*dir* Search *dir* for subdirectories *dev*name (*name* is the name of the device) for the *DESC* file before the default font directories */usr/local/share/groff/site–font*, */usr/local/share/groff/1.22.4/font*, and */usr/lib/font*.

–C Recognize **.GS** and **.GE** (and **.GF**) even when followed by a character other than space or newline.

grn Commands

Each input line between **.GS** and **.GE** may have one *grn* command. Commands consist of one or two strings separated by white space, the first string being the command and the second its operand. Commands may be upper or lower case and abbreviated down to one character.

Commands that affect a picture's environment (those listed before **default**, see below) are only in effect for the current picture: The environment is reinitialized to the defaults at the start of the next picture. The commands are as follows:

1 *N*

2 *N*

3 *N*

4 *N* Set *gremlin*'s text size number 1 (2, 3, or 4) to *N* points. The default is 12 (16, 24, and 36, respectively).

roman *f*

italics *f*

bold *f*

special *f*

Set the roman (italics, bold, or special) font to *troff*'s font *f* (either a name or number). The default is R (I, B, and S, respectively).

I *f*

stipple *f*

Set the stipple font to *troff*'s stipple font *f* (name or number). The command **stipple** may be abbreviated down as far as 'st' (to avoid confusion with **special**). There is*no* def ault for stipples (unless one is set by the default command), and it is invalid to include a *gremlin* picture with polygons without specifying a stipple font.

x *N*

scale *N*

Magnify the picture (in addition to any default magnification) by *N*, a floating point number larger than zero. The command**scale** ma y be abbreviated down to 'sc'.

narrow *N*

medium *N*

thick *N*

Set the thickness of *gremlin*'s narrow (medium and thick, respectively) lines to *N* times 0.15pt (this value can be changed at compile time). The default is 1.0 (3.0 and 5.0, respectively), which corresponds to 0.15pt (0.45pt and 0.75pt, respectively). A thickness value of zero selects the smallest available line thickness. Negative values cause the line thickness to be proportional to the current point size.

pointscale *<off/on>*

Scale text to match the picture. Gremlin text is usually printed in the point size specified with the commands **1**, **2**, **3**, or **4**, regardless of any scaling factors in the picture. Setting**pointscale** will cause the point siz es to scale with the picture (within *troff*'s limitations, of course). An operand of anything but *off* will turn text scaling on.

default

Reset the picture environment defaults to the settings in the current picture. This is meant to be used as a global parameter setting mechanism at the

beginning of the *troff* input file, but can be used at any time to reset the default settings.

width *N*
Forces the picture to be *N* inches wide. This overrides any scaling factors present in the same picture. '**width** *0*' is ignored.

height *N*
Forces picture to be *N* inches high, overriding other scaling factors. If both 'width' and 'height' are specified the tighter constraint will determine the scale of the picture. **Height** and **width** commands are not saved with a **default** command. They will, however, affect point size scaling if that option is set.

file *name*
Get picture from *gremlin* file *name* located the current directory (or in the library directory; see the **–M** option above). If two **file** commands are given, the second one overrides the first. If *name* doesn't exist, an error message is reported and processing continues from the **.GE** line.

Notes about groff

Since *grn* is a preprocessor, it doesn't know about current indents, point sizes, margins, number registers, etc. Consequently, no *troff* input can be placed between the **.GS** and **.GE** requests. However, *gremlin* text is now processed by *troff*, so anything valid in a single line of *troff* input is valid in a line of *gremlin* text (barring '.' directives at the beginning of a line). Thus, it is possible to have equations within a *gremlin* figure by including in the *gremlin* file *eqn* expressions enclosed by previously defined delimiters (e.g., *$$*). When using*g rn* along with other preprocessors, it is best to run *tbl* before *grn*, *pic*, and/or *ideal* to avoid overworking *tbl*. *Eqn* should always be run last.

A picture is considered an entity, but that doesn't stop *troff* from trying to break it up if it falls off the end of a page. Placing the picture between 'keeps' in –me macros will ensure proper placement. *g rn* uses *troff*'s number registers **g1** through **g9** and sets registers **g1** and **g2** to the width and height of the *gremlin* figure (in device units) before entering the **.GS** request (this is for those who want to rewrite these macros).

Gremlin File Format

There exist two distinct *gremlin* file formats, the original format from the *AED* graphic terminal version, and the *SUN* or *X11* version. An extension to the *SUN*/*X11* version allowing reference points with negative coordinates is **not** compatible with the *AED* version. As long as a*g remlin* file does not contain negative coordinates, either format will be read correctly by either version of *gremlin* or *grn*. The other difference from *SUN*/*X11* format is the use of names for picture objects (e.g., POLYGON, CURVE) instead of numbers. Files representing the same picture are shown in Table 1 in each format.

sungremlinfile	gremlinfile
0 240.00 128.00	0 240.00 128.00
CENTCENT	2
240.00 128.00	240.00 128.00
185.00 120.00	185.00 120.00
240.00 120.00	240.00 120.00

296.00 120.00	296.00 120.00
*	–1.00 –1.00
2 3	2 3
10 A Triangle	10 A Triangle
POLYGON	6
224.00 416.00	224.00 416.00
96.00 160.00	96.00 160.00
384.00 160.00	384.00 160.00
*	–1.00 –1.00
5 1	5 1
0	0
–1	–1

Table 1. File examples

- The first line of each *gremlin* file contains either the string **gremlinfile** (*AED* version) or **sungremlinfile** (*SUN*/*X11*)
- The second line of the file contains an orientation, and **x** and **y** values for a positioning point, separated by spaces. The orientation, either**or 1**, is ignored b y the *SUN*/*X11* version. **means that** *gremlin* will display things in horizontal format (drawing area wider than it is tall, with menu across top). **1** means that *gremlin* will display things in vertical format (drawing area taller than it is wide, with menu on left side). **x** and **y** are floating point v alues giving a positioning point to be used when this file is read into another file. The stuff on this line really isn't all that important; a value of "1 0.00 0.00" is suggested.
- The rest of the file consists of zero or more element specifications. After the last element specification is a line containing the string "–1".
- Lines longer than 127 characters are chopped to this limit.

Element Specifications

- The first line of each element contains a single decimal number giving the type of the element (*AED* version) or its ASCII name (*SUN*/*X11* version). See Table 2.

gremlin File Format — Object Type Specification

AED Number	*SUN*/*X11* Name	Description
0	BOTLEFT	bottom-left-justified text
1	BOTRIGHT	bottom-right-justified text
2	CENTCENT	center-justified text
3	VECTOR	vector
4	ARC	arc
5	CURVE	curve
6	POLYGON	polygon
7	BSPLINE	b-spline
8	BEZIER	Bézier
10	TOPLEFT	top-left-justified text
11	TOPCENT	top-center-justified text

12	TOPRIGHT	top-right-justified text
13	CENTLEFT	left-center-justified text
14	CENTRIGHT	right-center-justified text
15	BOTCENT	bottom-center-justified text

Table 2.
Type Specifications in *gremlin* Files

- After the object type comes a variable number of lines, each specifying a point used to display the element. Each line contains an x-coordinate and a y-coordinate in floating point format, separated by spaces. The list of points is terminated by a line containing the string "–1.0 –1.0" (*AED* version) or a single asterisk, "*" (*Sun*/*X11* version).
- After the points comes a line containing two decimal values, giving the brush and size for the element. The brush determines the style in which things are drawn. For vectors, arcs, and curves there are six valid brush values:

  1 — thin dotted lines
  2 — thin dot-dashed lines
  3 — thick solid lines
  4 — thin dashed lines
  5 — thin solid lines
  6 — medium solid lines

  For polygons, one more value, 0, is valid. It specifies a polygon with an invisible border. For text, the brush selects a font as follows:

  1 — roman (R font in groff)
  2 — italics (I font in groff)
  3 — bold (B font in groff)
  4 — special (S font in groff)

  If you're using *grn* to run your pictures through *groff*, the font is really just a starting font: The text string can contain formatting sequences like "\fI" or "\d" which may change the font (as well as do many other things). For text, the size field is a decimal value between 1 and 4. It selects the size of the font in which the text will be drawn. For polygons, this size field is interpreted as a stipple number to fill the polygon with. The number is used to index into a stipple font at print time.
- The last line of each element contains a decimal number and a string of characters, separated by a single space. The number is a count of the number of characters in the string. This information is only used for text elements, and contains the text string. There can be spaces inside the text. For arcs, curves, and vectors, this line of the element contains the string "0".

Notes on Coordinates

gremlin was designed for *AED*s, and its coordinates reflect the *AED* coordinate space. For vertical pictures, x-values range 116 to 511, and y-values from 0 to 483. For horizontal pictures, x-values range from 0 to 511 and y-values range from 0 to 367. Although you needn't absolutely stick to this range, you'll get best results if you at least stay in this vicinity. Also, point lists are terminated by a point of (–1, –1), so

you shouldn't ever use negative coordinates. *gremlin* writes out coordinates using format "%f1.2"; it's probably a good idea to use the same format if you want to modify the *grn* code.

Notes on Sun/X11 Coordinates

There is no longer a restriction on the range of coordinates used to create objects in the *SUN*/*X11* version of *gremlin*. However, files with negative coordinates **will** cause problems if displayed on the *AED*.

Files

*/usr/local/share/groff/1.22.4/font/dev*name*/DESC*
Device description file for device *name*.

Authors

David Slattengren and Barry Roitblat wrote the original Berkeley *grn*. Daniel Senderowicz and Werner Lemberg modified it for *groff*.

See Also

gremlin(1), **groff**(1), **pic**(1), **ideal**(1)

6.5. grap

A free implementation of grap, written by Ted Faber, is available as an extra package from the following address:

http://www.lunabase.org/~faber/Vault/software/grap/

6.6. gchem

6.6.1. Invoking gchem

Name

chem – groff preprocessor for producing chemical structure diagrams

Synopsis

chem [**––**] [*filespec* . . .]
chem –h
chem ––help
chem –v
chem ––version

Description

chem produces chemical structure diagrams. Today's version is best suited for organic chemistry (bonds, rings). The **chem** program is a **groff** preprocessor like **eqn**, **pic**, **tbl**, etc. It generates *pic* output such that all *chem* parts are translated into diagrams of the *pic* language. A*filespec* argument is either a file name of an e xisting file or a minus character **–**, meaning standard input. If no argument is specified then standard input is taken automatically. **–h** and **––help** display a usage message, whereas **–v** and **––version** display version information; all exit.

The program **chem** originates from the Perl source file *chem.pl*. It tells**pic** to include a copy of the macro file *chem.pic*. Moreover the *groff* source file *pic.tmac* is loaded. In a style reminiscent of *eqn* and *pic*, the *chem* diagrams are written in a special language.

A set of *chem* lines looks like this

.cstart
chem data
.cend
Lines containing the keywords
.cstart
and
.cend
start and end the input for
chem,
respectively.
In
pic
context, i.e., after the call of
.PS,

chem
input can optionally be started by the line
begin chem
and ended by the line with the single word
end
instead.

Anything outside these initialization lines is copied through without modification; all data between the initialization lines is converted into *pic* commands to draw the diagram. As an example,

```
.cstart
CH3
bond
CH3
.cend
```

prints two **CH3** groups with a bond between them. If you want to create just **groff** output, you must run **chem** followed by **groff** with the option **–p** for the activation of **pic**:

chem [*file* . . .] | **groff –p** . . .

The Language

The *chem* input language is rather small. It provides rings of several styles and a way to glue them together as desired, bonds of several styles, moieties (e.g., **C**, **NH3**, . . ., and strings.

Setting variables

There are some variables that can be set by commands. Such commands have two possible forms, either

variable value

or

variable **=** *value*

This sets the given *variable* to the argument *value*. If more arguments are given only the last argument is taken, all other arguments are ignored.

There are only a few variables to be set by these commands:

textht *arg*
Set the height of the text to *arg*; default is 0.16.

cwid *arg*
Set the character width to *arg*; default is 0.12.

db *arg*
Set the bond length to *arg*; default is 0.2.

size *arg*
Scale the diagram to make it look plausible at point size *arg*; default is 10 point.

Bonds

This

bond [*direction*] [*length n*] [**from** *Name*|*picstuff*]

draws a single bond in direction from nearest corner of *Name*. **bond** can also be **double bond**, **front bond**, **back bond**, etc. (We will get back to *Name* soon.)

direction is the angle in degrees (0 up, positive clockwise) or a direction word like **up**, **down**, **sw** (= southwest), etc. If no direction is specified, the bond goes in the current direction (usually that of the last bond). Normally the bond begins at the last object placed; this can be changed by naming a **from** place. For instance, to make a simple alkyl chain:

CH3
bond (this one goes right from the CH3)
C (at the right end of the bond)
double bond up (from the C)
O (at the end of the double bond)
bond right from C
CH3

A length in inches may be specified to override the default length. Other *pic* commands can be tacked on to the end of a bond command, to created dotted or dashed bonds or to specify a **to** place.

Rings

There are lots of rings, but only 5 and 6-sided rings get much support. **ring** b y itself is a 6-sided ring; **benzene** is the benzene ring with a circle inside. **aromatic** puts a circle into any kind of ring.

ring [**pointing** (**up**|**right**|**left**|**down**)] [**aromatic**] [**put Mol at** *n*] [-
double *i*,*j k*,*l . . .* [*picstuff*]

The vertices of a ring are numbered 1, 2, . . . from the vertex that points in the natural compass direction. So for a hexagonal ring with the point at the top, the top vertex is 1, while if the ring has a point at the east side, that is vertex 1. This is expressed as

R1: ring pointing up
R2: ring pointing right

The ring vertices are named **.V1**, . . ., **.V***n*, with **.V1** in the pointing direction. So the corners of **R1** are **R1.V1** (the *top*), **R1.V2**, **R1.V3**, **R1.V4** (the *bottom*), etc., whereas for **R2**, **R2.V1** is the rightmost vertex and **R2.V4** the leftmost. These vertex names are used for connecting bonds or other rings. For example,

R1: benzene pointing right
R2: benzene pointing right with .V6 at R1.V2
creates two benzene rings connected along a side.

Interior double bonds are specified as **double** *n1*,*n2 n3*,*n4 . . .*; each number pair adds an interior bond. So the alternate form of a benzene ring is

ring double 1,2 3,4 5,6 Heterocycles (rings with something other than carbon at a vertex) are written as **put** *X* **at** *V*, as in

R: ring put N at 1 put O at 2

In this heterocycle, **R.N** and **R.O** become synonyms for **R.V1** and **R.V2**. There are two 5-sided rings. **ring5** is pentagonal with a side that matches the 6-sided ring; it has four natural directions. A**flatring** is a 5-sided ring created b y chopping one corner of a 6-sided ring so that it exactly matches the 6-sided rings.

The description of a ring has to fit on a single line.

Moieties and strings

A moiety is a string of characters beginning with a capital letter, such as N(C2H5)2. Numbers are converted to subscripts (unless they appear to be fractional values, as in N2.5H). The name of a moiety is determined from the moiety after special characters have been stripped out: e.g., N(C2H5)2) has the name NC2H52. Moieties can be specified in two kinds. Normally a moiety is placed right after the last thing mentioned, separated by a semicolon surrounded by spaces, e.g.,

B1: bond ; OH

Here the moiety is **OH**; it is set after a bond. As the second kind a moiety can be positioned as the first word in a *pic*-like command, e.g.,

CH3 at C + (0.5,0.5)

Here the moiety is **CH3**. It is placed at a position relative to **C**, a moiety used earlier in the chemical structure. So moiety names can be specified as*chem* positions e verywhere in the *chem* code. Beneath their printing moieties are names for places.

The moiety **BP** is special. It is not printed but just serves as a mark to be referred to in later *chem* commands. For example,

bond ; BP sets a mark at the end of the bond. This can be used then for specifying a place. The name**BP** is deriv ed from *branch point* (i.e., line crossing).

A string within double quotes **"** is interpreted as a part of a *chem* command. It represents a string that should be printed (without the quotes). Text within quotes **"**. . .**"** is treated more or less like a moiety except that no changes are made to the quoted part.

Names

In the alkyl chain above, notice that the carbon atom **C** was used both to draw something and as the name for a place. A moiety always defines a name for a place; you can use your own names for places instead, and indeed, for rings you will have to. A name is just

*Name***:** . . . *Name* is often the name of a moiety like **CH3**, but it need not to be. Any name that begins with a capital letter and which contains only letters and numbers is valid:

First: " bond"

bond 30 from First

Miscellaneous

The specific construction

bond . . . **; moiety**

is equivalent to

bond
moiety
Otherwise, each item has to be on a separate line (and only one line).
Note that there must be whitespace after the semicolon which separates the commands.

A period character **.** or a single quote **'** in the first column of a line signals a *troff* command, which is copied through as-is. A line whose first non-blank character is a hash character (**#**) is treated as a comment and thus ignored. However, hash characters within a word are kept.

A line whose first word is **pic** is copied through as-is after the word **pic** has been removed. The command

size *n*

scales the diagram to make it look plausible at point size *n* (default is 10 point). Anything else is assumed to be *pic* code, which is copied through with a label.

Since **chem** is a **pic** preprocessor, it is possible to include *pic* statements in the middle of a diagram to draw things not provided for by *chem* itself. Such*pic* statements should be included in *chem* code by adding **pic** as the first word of this line for clarity. The following *pic* commands are accepted as *chem* commands, so no **pic** command word is needed:

define Start the definition of *pic* macro within *chem*.

[Start a block composite.

] End a block composite.

{ Start a macro definition block.

} End a macro definition block.

The macro names from **define** statements are stored and their call is accepted as a *chem* command as well.

Wish list

This TODO list was collected by Brian Kernighan.

Error checking is minimal; errors are usually detected and reported in an oblique fashion by *pic*. There is no library or file inclusion mechanism, and there is no shorthand for repetitive structures.

The extension mechanism is to create *pic* macros, but these are tricky to get right and don't have all the properties of built-in objects. There is no in-line chemistry yet (e.g., analogous to the **$**. . .**$** construct of *eqn*).

There is no way to control entry point for bonds on groups. Normally a bond connects to the carbon atom if entering from the top or bottom and otherwise to the nearest corner. Bonds from substituted atoms on heterocycles do not join at the proper place without adding a bit of *pic*.

There is no decent primitive for brackets. Text (quoted strings) doesn't work very well.

A squiggle bond is needed.

Files

/usr/local/share/groff/1.22.4/pic/chem.pic
A collection of *pic* macros needed by **chem**.

/usr/local/share/groff/1.22.4/tmac/pic.tmac
A macro file which redefines **.PS** and **.PE** to center *pic* diagrams.

/usr/local/share/doc/groff–1.22.4/examples/chem/.chem*
Example files for *chem*.

/usr/local/share/doc/groff–1.22.4/examples/chem/122/.chem*
Example files from the *chem* article by its authors, “CHEM—A Program for Typesetting Chemical Structure Diagrams: User Manual” (CSTR #122).

Authors

The GNU version of *chem* was written by Bernd Warken It is based on the documentation of Brian Kernighan’s original *awk* version of *chem*.

See Also

“CHEM—A Program for Typesetting Chemical Diagrams: User Manual” by Jon L. Bentley, Lynn W. Jelinski, and Brian W. Kernighan, AT&T Bell Laboratories Computing Science Techical Report No. 122 (CSTR #122) **groff**(1), **pic**(1)

6.7. grefer

6.7.1. Invoking grefer

Name

refer – preprocess bibliographic references for groff

Synopsis

refer [**–benCPRS**] [**–a***n*] [**–c***fields*] [**–f***n*] [**–i***fields*] [**–k***field*] [**–l***m,n*] [**–p***filename*] [**–s***fields*] [**–t***n*] **–B** *field***.***macro* [*file* . . .]
refer ––help
refer –v
refer ––version

Description

This file documents the GNU version of **refer**, which is part of the groff document formatting system. **ref er** copies the contents of *filename* . . . to the standard output, except that lines between **.[** and **.]** are interpreted as citations, and lines between **.R1** and **.R2** are interpreted as commands about how citations are to be processed.

Each citation specifies a reference. The citation can specify a reference that is contained in a bibliographic database by giving a set of keywords that only that reference contains. Alternatively it can specify a reference by supplying a database record in the citation. A combination of these alternatives is also possible. For each citation, **refer** can produce a mark in the text. This mark consists of some label which can be separated from the text and from other labels in various ways. For each reference it also outputs **groff** commands that can be used by a macro package to produce a formatted reference for each citation. The output of **refer** must therefore be processed using a suitable macro package, such as *ms*, *man*, *me*, or *mm*. The commands to format a citation's reference can be output immediately after the citation, or the references may be accumulated, and the commands output at some later point. If the references are accumulated, then multiple citations of the same reference will produce a single formatted reference.

The interpretation of lines between **.R1** and **.R2** as commands is a new feature of GNU **refer**. Documents making use of this feature can still be processed by Unix refer just by adding the lines

```
.de R1
.ig R2
..
```

to the beginning of the document. This will cause **troff** to ignore everything between **.R1** and **.R2**. The effect of some commands can also be achieved by options. These options are supported mainly for compatibility with Unix refer. It is usually more convenient to use commands.

refer generates **.lf** lines so that filenames and line numbers in messages produced by commands that read **refer** output will be correct; it also interprets lines beginning with **.lf** so that filenames and line numbers in the messages and **.lf** lines that it produces will be accurate even if the input has been preprocessed by a command such as **soelim**(1).

Options

−−help displays a usage message, while **−v** and **−−version** show version information; all exit afterward. Most options are equivalent to commands (for a description of these commands, see subsection “Commands” below).

−b **no-label-in-text; no-label-in-reference**

−e **accumulate**

−n **no-default-database**

−C **compatible**

−P **move-punctuation**

−S **label "(A.n|Q) ’, ’ (D.y|D)"; bracket-label " (") "; "**

−a*n* **reverse A***n*

−c*fields*
capitalize *fields*

−f*n* **label %***n*

−i*fields*
search-ignore *fields*

−k **label L~%a**

−k*field*
label *field*~**%a**

−l **label A.nD.y%a**

−l*m* **label A.n***m***D.y%a**

−l,*n* **label A.nD.y−***n***%a**

−l*m*,*n* **label A.n***m***D.y−***n***%a**

−p*filename*
database *filename*

−s*spec*
sort *spec*

−t*n* **search-truncate** *n*

These options are equivalent to the following commands with the addition that the filenames specified on the command line are processed as if they were arguments to the **bibliography** command instead of in the normal way:

−B **annotate X AP; no-label-in-reference**

−B*field***.***macro*
annotate *field macro***; no-label-in-reference** The following options have no equivalent commands:

−R Don’t recognize lines beginning with **.R1**/**.R2**.

Usage

Bibliographic databases

The bibliographic database is a text file consisting of records separated by one or more blank lines. Within each record fields start with a**%** at the beginning of a line .

Each field has a one character name that immediately follows the **%**. It is best to use only upper and lower case letters for the names of fields. The name of the field should be followed by exactly one space, and then by the contents of the field. Empty fields are ignored. The conventional meaning of each field is as follows:

%A The name of an author. If the name contains a title such as**Jr .** at the end, it should be separated from the last name by a comma. There can be multiple occurrences of the **%A** field. The order is significant. It is a good idea always to supply an **%A** field or a **%Q** field.

%B For an article that is part of a book, the title of the book.

%C The place (city) of publication.

%D The date of publication. The year should be specified in full. If the month is specified, the name rather than the number of the month should be used, but only the first three letters are required. It is a good idea always to supply a **%D** field; if the date is unknown, a value such as **in press** or **unknown** can be used.

%E For an article that is part of a book, the name of an editor of the book. Where the work has editors and no authors, the names of the editors should be given as **%A** fields and **, (ed)** or **, (eds)** should be appended to the last author.

%G US Government ordering number.

%I The publisher (issuer).

%J For an article in a journal, the name of the journal.

%K Keywords to be used for searching.

%L Label.

%N Journal issue number.

%O Other information. This is usually printed at the end of the reference.

%P Page number. A range of pages can be specified as *m–n*.

%Q The name of the author, if the author is not a person. This will only be used if there are no **%A** fields. There can only be one**%Q** field.

%R Technical report number.

%S Series name.

%T Title. For an article in a book or journal, this should be the title of the article.

%V Volume number of the journal or book.

%X Annotation.

For all fields except **%A** and **%E**, if there is more than one occurrence of a particular field in a record, only the last such field will be used. If accent strings are used, they should follow the character to be accented. This means that the **AM** macro must be used with the **–ms** macros. Accent strings should not be quoted: use one\ r ather than two.

Citations

The format of a citation is

.[*opening-text*
flags keywords

fields
.]*closing-text*

The *opening-text*, *closing-text*, and *flags* components are optional. Only one of the *keywords* and *fields* components need be specified. The *keywords* component says to search the bibliographic databases for a reference that contains all the words in *keywords*. It is an error if more than one reference if found.

The *fields* components specifies additional fields to replace or supplement those specified in the reference. When references are being accumulated and the *keywords* component is non-empty, then additional fields should be specified only on the first occasion that a particular reference is cited, and will apply to all citations of that reference. The*opening-te xt* and *closing-text* component specifies strings to be used to bracket the label instead of the strings specified in the **bracket-label** command. If either of these components is non-empty, the strings specified in the **bracket-label** command will not be used; this behaviour can be altered using the **[** and **]** flags. Note that leading and trailing spaces are significant for these components.

The *flags* component is a list of non-alphanumeric characters each of which modifies the treatment of this particular citation. Unix refer will treat these flags as part of the keywords and so will ignore them since they are non-alphanumeric. The following flags are currently recognized:

This says to use the label specified by the **short-label** command, instead of that specified by the **label** command. If no short label has been specified, the normal label will be used. Typically the short label is used with author-date labels and consists of only the date and possibly a disambiguating letter; the **#** is supposed to be suggestive of a numeric type of label.

[Precede *opening-text* with the first string specified in the **bracket-label** command.

] Follow *closing-text* with the second string specified in the **bracket-label** command. One advantages of using the **[** and **]** flags rather than including the brackets in *opening-text* and *closing-text* is that you can change the style of bracket used in the document just by changing the **bracket-label** command. Another advantage is that sorting and merging of citations will not necessarily be inhibited if the flags are used.

If a label is to be inserted into the text, it will be attached to the line preceding the **.[** line. If there is no such line, then an extra line will be inserted before the **.[** line and a warning will be given. There is no special notation for making a citation to multiple references. Just use a sequence of citations, one for each reference. Don't put anything between the citations. The labels for all the citations will be attached to the line preceding the first citation. The labels may also be sorted or merged. See the description of the **<>** label expression, and of the **sort-adjacent-labels** and **abbreviate-label-ranges** command. A label will not be merged if its citation has a non-empty *opening-text* or *closing-text*. However, the labels for a citation using the **]** flag and without any *closing-text* immediately followed by a citation using the **[** flag and without any *opening-text* may be sorted and merged even though the first citation's *opening-text* or the second citation's *closing-text* is non-empty. (If you wish to prevent this just make the first citation's *closing-text* **&**.)

Commands

Commands are contained between lines starting with **.R1** and **.R2**. Recognition of these lines can be prevented by the **–R** option. When a **.R1** line is recognized any accumulated references are flushed out. Neither **.R1** nor **.R2** lines, nor anything between them is output.

Commands are separated by newlines or **;**s. **#** introduces a comment that extends to the end of the line (but does not conceal the newline). Each command is broken up into words. Words are separated by spaces or tabs. A word that begins with **"** extends to the next **"** that is not followed by another **"**. If there is no such**"** the w ord extends to the end of the line. Pairs of **"** in a word beginning with **"** collapse to a single **"**. Neither**#** nor **;** are recogniz ed inside **"**s. A line can be continued by ending it with \; this works everywhere except after a **#**. Each command*name* that is mark ed with * has an associated negative command **no-***name* that undoes the effect of *name*. For example, the **no-sort** command specifies that references should not be sorted. The negative commands take no arguments.

In the following description each argument must be a single word; *field* is used for a single upper or lower case letter naming a field; *fields* is used for a sequence of such letters; *m* and *n* are used for a non-negative numbers; *string* is used for an arbitrary string; *filename* is used for the name of a file.

abbreviate* *fields string1 string2 string3 string4*

Abbreviate the first names of *fields*. An initial letter will be separated from another initial letter by *string1*, from the last name by *string2*, and from anything else (such as a **von** or **de**) by *string3*. These default to a period followed by a space. In a hyphenated first name, the initial of the first part of the name will be separated from the hyphen by *string4*; this defaults to a period. No attempt is made to handle any ambiguities that might result from abbreviation. Names are abbreviated before sorting and before label construction.

abbreviate-label-ranges* *string*

Three or more adjacent labels that refer to consecutive references will be abbreviated to a label consisting of the first label, followed by *string* followed by the last label. This is mainly useful with numeric labels. If*string* is omitted it defaults to **–**.

accumulate*

Accumulate references instead of writing out each reference as it is encountered. Accumulated references will be written out whenever a reference of the form

.[
$LIST$
.] is encountered, after all input files have been processed, and whenever **.R1** line is recognized.

annotate* *field string*

field is an annotation; print it at the end of the reference as a paragraph preceded by the line

.*string*

If *string* is omitted it will default to **AP**; if *field* is also omitted it will default to **X**. Only one field can be an annotation.

articles *string* . . .
string . . . are definite or indefinite articles, and should be ignored at the beginning of **T** fields when sorting. Initially, **the**, **a** and **an** are recognized as articles.

bibliography *filename* . . .
Write out all the references contained in the bibliographic databases *filename* . . . This command should come last in a**.R1**/**.R2** b lock.

bracket-label *string1 string2 string3*
In the text, bracket each label with *string1* and *string2*. An occurrence of *string2* immediately followed by *string1* will be turned into *string3*. The default behaviour is

bracket-label *([. *(.] ", "

capitalize *fields*
Convert *fields* to caps and small caps.

compatible*
Recognize **.R1** and **.R2** even when followed by a character other than space or newline.

database *filename* . . .
Search the bibliographic databases *filename* . . . For each *filename* if an index *filename***.i** created by **indxbib**(1) exists, then it will be searched instead; each index can cover multiple databases.

date-as-label* *string*
string is a label expression that specifies a string with which to replace the **D** field after constructing the label. See subsection “Label expressions” below for a description of label expressions. This command is useful if you do not want explicit labels in the reference list, but instead want to handle any necessary disambiguation by qualifying the date in some way. The label used in the text would typically be some combination of the author and date. In most cases you should also use the **no-label-in-reference** command. For example,

date-as-label D.+yD.y%a*D.-y would attach a disambiguating letter to the year part of the **D** field in the reference.

default-database*
The default database should be searched. This is the default behaviour, so the negative version of this command is more useful. **ref er** determines whether the default database should be searched on the first occasion that it needs to do a search. Thus a**no-default-database** command m ust be given before then, in order to be effective.

discard* *fields*
When the reference is read, *fields* should be discarded; no string definitions for *fields* will be output. Initially, *fields* are **XYZ**.

et-al* *string m n*
Control use of **et al** in the evaluation of @ expressions in label expressions. If the number of authors needed to make the author sequence unambiguous is *u* and the total number of authors is *t* then the last $t - u$ authors will be replaced

by *string* provided that *t – u* is not less than *m* and *t* is not less than *n*. The default behaviour is

et-al " et al" 2 3

include *filename*
Include *filename* and interpret the contents as commands.

join-authors *string1 string2 string3*
This says how authors should be joined together. When there are exactly two authors, they will be joined with *string1*. When there are more than two authors, all but the last two will be joined with *string2*, and the last two authors will be joined with *string3*. If*string3* is omitted, it will def ault to *string1*; if *string2* is also omitted it will also default to *string1*. For example,

join-authors " and " ", " ", and "

will restore the default method for joining authors.

label-in-reference*
When outputting the reference, define the string **[F** to be the reference's label. This is the default behaviour; so the negative version of this command is more useful.

label-in-text*
For each reference output a label in the text. The label will be separated from the surrounding text as described in the **bracket-label** command. This is the default behaviour; so the negative version of this command is more useful.

label *string*
string is a label expression describing how to label each reference.

separate-label-second-parts *string*
When merging two-part labels, separate the second part of the second label from the first label with *string*. See the description of the**<>** label e xpression.

move-punctuation*
In the text, move any punctuation at the end of line past the label. It is usually a good idea to give this command unless you are using superscripted numbers as labels.

reverse* *string*
Reverse the fields whose names are in *string*. Each field name can be followed by a number which says how many such fields should be reversed. If no number is given for a field, all such fields will be reversed.

search-ignore* *fields*
While searching for keys in databases for which no index exists, ignore the contents of *fields*. Initially, fields **XYZ** are ignored.

search-truncate* *n*
Only require the first *n* characters of keys to be given. In effect when searching for a given key words in the database are truncated to the maximum of *n* and the length of the key. Initially*n* is 6.

short-label* *string*
string is a label expression that specifies an alternative (usually shorter) style of label. This is used when the **#** flag is given in the citation. When using

author-date style labels, the identity of the author or authors is sometimes clear from the context, and so it may be desirable to omit the author or authors from the label. The **short-label** command will typically be used to specify a label containing just a date and possibly a disambiguating letter.

sort* *string*
Sort references according to **string**. References will automatically be accumulated. *string* should be a list of field names, each followed by a number, indicating how many fields with the name should be used for sorting. **+** can be used to indicate that all the fields with the name should be used. Also **.** can be used to indicate the references should be sorted using the (tentative) label. (Subsection "Label expressions" below describes the concept of a tentative label.)

sort-adjacent-labels*
Sort labels that are adjacent in the text according to their position in the reference list. This command should usually be given if the **abbreviate-label-ranges** command has been given, or if the label expression contains a **<>** expression. This will have no effect unless references are being accumulated.

Label expressions

Label expressions can be evaluated both normally and tentatively. The result of normal evaluation is used for output. The result of tentative evaluation, called the *tentative label*, is used to gather the information that normal evaluation needs to disambiguate the label. Label expressions specified by the **date-as-label** and **short-label** commands are not evaluated tentatively. Normal and tentative evaluation are the same for all types of expression other than @, *****, and **%** expressions. The description below applies to normal evaluation, except where otherwise specified.

field

field n
The *n*-th part of *field*. If *n* is omitted, it defaults to 1.

'*string*'
The characters in *string* literally.

@ All the authors joined as specified by the **join-authors** command. The whole of each author's name will be used. However, if the references are sorted by author (that is the sort specification starts with **A**), then authors last names will be used instead, provided that this does not introduce ambiguity, and also an initial subsequence of the authors may be used instead of all the authors, again provided that this does not introduce ambiguity. The use of only the last name for the i-th author of some reference is considered to be ambiguous if there is some other reference, such that the first $i-1$ authors of the references are the same, the i-th authors are not the same, but the i-th authors last names are the same. A proper initial subsequence of the sequence of authors for some reference is considered to be ambiguous if there is a reference with some other sequence of authors which also has that subsequence as a proper initial subsequence. When an initial subsequence of authors is used, the remaining authors are replaced by the string specified by the **et-al** command; this command may also specify additional requirements that must be met before an initial subsequence can be used. @ tentativ ely evaluates to a canonical representation of the authors, such that authors that compare equally for sorting purpose will

have the same representation.

%*n*

%a

%A

%i

%I The serial number of the reference formatted according to the character following the **%**. The serial number of a reference is 1 plus the number of earlier references with same tentative label as this reference. These expressions tentatively evaluate to an empty string.

*expr******
If there is another reference with the same tentative label as this reference, then *expr*, otherwise an empty string. It tentatively evaluates to an empty string.

expr n

*expr***–***n*
The first () or last (**–**) *n* upper or lower case letters or digits of *expr*. Troff special characters (such as **(’a**) count as a single letter. Accent strings are retained but do not count towards the total.

*expr***.l**
expr converted to lowercase.

*expr***.u**
expr converted to uppercase.

*expr***.c**
expr converted to caps and small caps.

*expr***.r**
expr reversed so that the last name is first.

*expr***.a**
expr with first names abbreviated. Note that fields specified in the**abbre viate** command are abbreviated before any labels are evaluated. Thus**.a** is useful only when you want a field to be abbreviated in a label but not in a reference.

*expr***.y**
The year part of *expr*.

*expr***.**
The part of *expr* before the year, or the whole of *expr* if it does not contain a year.

*expr***.–y**
The part of *expr* after the year, or an empty string if *expr* does not contain a year.

*expr***.n**
The last name part of *expr*.

expr1~*expr2*
expr1 except that if the last character of *expr1* is **–** then it will be replaced by *expr2*.

expr1 expr2
The concatenation of *expr1* and *expr2*.

expr1|*expr2*
If *expr1* is non-empty then *expr1* otherwise *expr2*.

*expr1***&***expr2*
If *expr1* is non-empty then *expr2* otherwise an empty string.

*expr1***?***expr2***:***expr3*
If *expr1* is non-empty then *expr2* otherwise *expr3*.

<*expr***>**
The label is in two parts, which are separated by *expr*. Two adjacent two-part labels which have the same first part will be merged by appending the second part of the second label onto the first label separated by the string specified in the **separate-label-second-parts** command (initially, a comma followed by a space); the resulting label will also be a two-part label with the same first part as before merging, and so additional labels can be merged into it. Note that it is permissible for the first part to be empty; this maybe desirable for expressions used in the **short-label** command.

(*expr***)**
The same as *expr*. Used for grouping. The above expressions are listed in order of precedence (highest first); **&** and | have the same precedence.

Macro interface

Each reference starts with a call to the macro **]-**. The string**[F** will be defined to be the label for this reference, unless the **no-label-in-reference** command has been given. There then follows a series of string definitions, one for each field: string **[***X* corresponds to field *X*. The number register **[P** is set to 1 if the **P** field contains a range of pages. The**[T**, **[A** and **[O** n umber registers are set to 1 according as the **T**, **A** and **O** fields end with one of the characters **.?!**. The**[E** n umber register will be set to 1 if the **[E** string contains more than one name. The reference is followed by a call to the **][** macro. The first argument to this macro gives a number representing the type of the reference. If a reference contains a **J** field, it will be classified as type 1, otherwise if it contains a **B** field, it will type 3, otherwise if it contains a **G** or **R** field it will be type 4, otherwise if it contains an **I** field it will be type 2, otherwise it will be type 0. The second argument is a symbolic name for the type: **other**, **journal-article**, **book**, **article-in-book** or **tech-report**. Groups of references that have been accumulated or are produced by the **bibliography** command are preceded by a call to the **]<** macro and followed by a call to the **]>** macro.

Environment

REFER
If set, overrides the default database.

Files

/usr/dict/papers/Ind
Default database.

file.*i* Index files.

refer uses temporary files. See the**gr off**(1) man page for details where such files are created.

Bugs

In label expressions, **<>** expressions are ignored inside **.***char* expressions.

See Also

“Some Applications of Inverted Indexes on the Unix System”; Computing Science Technical Report #69; M. E. Lesk; AT&T Bell Laboratories; 1978.

6.8. gsoelim

6.8.1. Invoking gsoelim

Name

soelim – interpret source (.so) requests in troff input

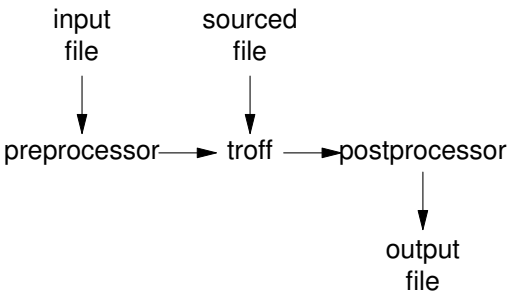
Synopsis

soelim [**–Crt**] [**–I** *dir*] [*input-file* . . .]
soelim ––help
soelim –v
soelim ––version

Description

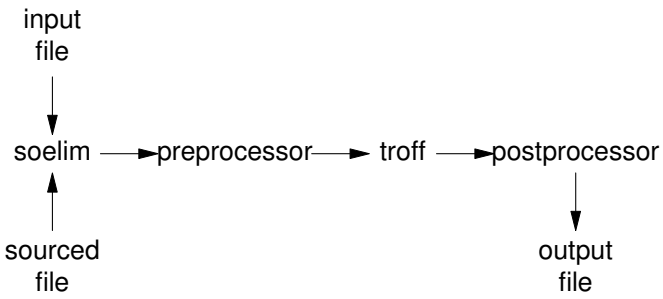
soelim replaces lines of the form "**.so** *macro-file*" within each*input-file* with the contents of *macro-file*, recursively. It is useful if files included with**.so** need to be preprocessed. Output is written to the standard output stream. Normally, *soelim* should be invoked with *groff*'s **–s** option. In the absence of *input-file* arguments, *soelim* reads the standard input stream. To embed a backslash "\" in the name of a*macro-file*, write "\\" or "**e**". To embed a space, write "\ " (backslash followed by a space). Any other escape sequence in *macro-file*, including "**[rs]**", makes *soelim* ignore the whole line.

There must be no whitespace between the leading dot and the two characters "s" and "o". Otherwise, only *groff* will interpret the **.so** request; *soelim* will ignore it (but see the **–C** option below). The normal processing sequence of*g roff* is as follows.



That is, files sourced with **.so** are normally read *only* by *troff* (the actual formatter). *soelim* is *not* required for *troff* to source files.

If a file to be sourced should also be preprocessed, it must already be read *before* the input file passes through the preprocessor. *soelim* handles this.



Options

––help displays a usage message, while **–v** and **––version** show version information; all exit afterward.

–C Recognize **.so** even when followed by a character other than space or newline.

–I *dir* Add the directory *dir* to the search path for macro files (both those on the command line and those named in **.so** requests). The search path is initialized with the current directory. This option may be specified more than once; the directories are then searched in the order specified (but before the current directory). If you want to search the current directory before other directories, add "**–I .**" at the appropriate place.

No directory search is performed for files with an absolute file name.

–r Do not add **.lf** requests (for general use, with non-*groff* files).

–t Emit TEX comment lines starting with "**%**" indicating the current file and line number, rather than **.lf** requests for the same purpose.

See Also

groff(1)

6.9. preconv

6.9.1. Invoking preconv

Name

preconv – prepare files for typesetting with GNU roff

Synopsis

preconv [**–dr**] [**–D***default-encoding*] [**–e***encoding*] [*file* . . .]
preconv –h
preconv ––help
preconv –v
preconv ––version

Description

preconv reads each *file*, converts its encoded characters to a form *groff*(1) can interpret, and sends the result to the standard output stream. Currently, this means that code points in the range 0–127 (in US-ASCII, ISO 8859, or Unicode) remain as-is and the remainder are converted to the *groff* special character form "**\[u***XXXX***]",** **where** *XXXX* is a hexadecimal number of four to six digits corresponding to a Unicode code point. By default, *preconv* also inserts a *roff* **.lf** request at the beginning of each *file*, identifying it for the benefit of later processing (including diagnostic messages); the **–r** option suppresses this behavior.

In typical usage scenarios, *preconv* need not be run directly; instead it should be invoked with the **–k** or **–K** options of *groff*. *preconv* tries to find the input encoding with the following algorithm, stopping at the first success.

1. If the input encoding has been explicitly specified with option **–e**, use it.
2. Check whether the input starts with a Unicode Byte Order Mark. If so, determine the encoding as UTF-8, UTF-16, or UTF-32 accordingly.
3. If the input stream is seekable, check the first and second input lines for a recognized GNU Emacs file-local variable identifying the character encoding, here referred to as the "coding tag" for brevity. If found, use it.
4. If the input stream is seekable, and if the *uchardet* library is available on the system, use it to try to infer the encoding of the file.
5. If the **–D** option specifies an encoding, use it.
6. Use the encoding specified by the current locale (*LC_CTYPE*), unless the locale is "C", "POSIX", or empty, in which case assume Latin-1 (ISO 8859-1).

Note that the coding tag and *uchardet* methods in the above procedure rely upon a seekable input stream; when *preconv* reads from a pipe, the stream is not seekable, and these detection methods are skipped. If character encoding detection of your input files is unreliable, arrange for one of the other methods to succeed by using *preconv*'s **–D** or **–e** options, or by configuring your locale appropriately. Furthermore, *groff* supports a *GROFF_ENCODING* environment variable which is equivalent to its option **–k**.

Coding tags

Text editors that support more than a single character encoding need tags within the input files to mark the file's encoding. While it is possible to guess the right input encoding with the help of heuristics that are reliable for a preponderance of natural language texts, they are not absolutely reliable. Heuristics can fail on inputs that are too short or don't represent a natural language. Consequently, *preconv* supports the coding tag convention (with some restrictions) used by GNU Emacs. These are indicated in specially-marked regions of an input file designated for "file-local variables".

preconv interprets the following syntax if it occurs in a *roff* comment in the first or second line of the input file. Both "\"" and "\#" comment forms are recognized, but the control (or non-breaking control) character must be the default and must begin the line. Similarly, the escape character must be the default.

`—*— [. . .;] coding: `*`encoding`*`[; . . .] —*—`

The only variable *preconv* interprets is "coding", which can take the values listed below.

The following list comprises all MIME "charset" parameter values recognized, case-insensitively, by *preconv*.

big5, cp1047, euc–jp, euc–kr, gb2312, iso–8859–1, iso–8859–2, iso–8859–5, iso–8859–7, iso–8859–9, iso–8859–13, iso–8859–15, koi8–r, us–ascii, utf–8, utf–16, utf–16be, utf–16le

In addition, the following list of other coding tags is recognized, each of which is mapped to an appropriate value from the list above.

ascii, chinese–big5, chinese–euc, chinese–iso–8bit, cn–big5, cn–gb, cn–gb–2312, cp878, csascii, csisolatin1, cyrillic–iso–8bit, cyrillic–koi8, euc–china, euc–cn, euc–japan, euc–japan–1990, euc–korea, greek–iso–8bit, iso–10646/utf8, iso–10646/utf–8, iso–latin–1, iso–latin–2, iso–latin–5, iso–latin–7, iso–latin–9, japanese–euc, japanese–iso–8bit, jis8, koi8, korean–euc, korean–iso–8bit, latin–0, latin1, latin–1, latin–2, latin–5, latin–7, latin–9, mule–utf–8, mule–utf–16, mule–utf–16be, mule–utf–16–be, mule–utf–16be–with–signature, mule–utf–16le, mule–utf–16–le, mule–utf–16le–with–signature, utf8, utf–16–be, utf–16–be–with–signature, utf–16be–with–signature, utf–16–le, utf–16–le–with–signature, utf–16le–with–signature

Trailing "–dos", "–unix", and "–mac" suffixes on coding tags (which indicate the end-of-line convention used in the file) are disregarded for the purpose of comparison with the above tags.

iconv support

preconv itself only supports three encodings: Latin-1, code page 1047, and UTF-8. If *iconv* support is configured at compile time and available at run time, all other encodings are passed to *iconv* library functions. The command "**preconv –v**" discloses whether *iconv* support is configured. The use of *iconv* means that characters in the input that encode invalid code points for that encoding may be dropped from the output stream or mapped to the Unicode replacement character (U+FFFD). Compare the following examples using the input "café" (note the "e" with an acute accent), which due to its short length challenges inference of the encoding used.

```
printf 'caf\351\n' | LC_ALL=en_US.UTF—8 preconv
printf 'caf\351\n' | preconv —e us—ascii
```

```
printf 'caf\351\n' | preconv —e latin—1
```

The fate of the accented “e” differs in each case. In the first,*uchardet* f ails to detect an encoding (though the library on your system may behave differently) and *preconv* falls back to the locale settings, where octal 351 starts an incomplete UTF-8 sequence and results in the Unicode replacement character. In the second, it is not a representable character in the declared input encoding of US-ASCII and is discarded by *iconv*. In the last, it is correctly detected and mapped.

Options

–h and **––help** display a usage message, while **–v** and **––version** show version information; all exit afterward.

–d Emit debugging messages to the standard error stream.

–D *default-encoding*
: Report *default-encoding* if all detection methods fail.

–e *encoding*
: Override detection procedure and assume *encoding*. This corresponds to *groff*’s “**–K** *encoding*” option.

–r Write files “raw”; do not add **.lf** requests.

See Also

groff(1), *iconv*(3), *locale*(7)

7. Output Devices

7.1. Special Characters

See Device and Font Files.

7.2. grotty

The postprocessor grotty translates the output from GNU troff into a form suitable for typewriter-like devices. It is fully documented on its man page, *grotty*(1).

7.2.1. Invoking grotty

The postprocessor grotty accepts the following command-line options:

-b Do not overstrike bold glyphs. Ignored if-c isn't used.

-B Do not underline bold-italic glyphs. Ignored if-c isn't used.

-c Use overprint and disable colours for printing on legacy Teletype printers (see below).

-d Do not render lines (that is, ignore all \D escapes).

-f Use form feed control characters in the output.

-F*dir* Put the directory *dir*/devname in front of the search path for the font and device description files, given the target device *name*.

-h Use horizontal tabs for sequences of 8 space characters.

-i Request italic glyphs from the terminal. Ignored if -c is active.

-o Do not overstrike.

-r Highlight italic glyphs. Ignored if-c is activ e.

-u Do not underline italic glyphs. Ignored if-c isn't used.

-U Do not overstrike bold-italic glyphs. Ignored if-c isn't used.

-v Print the version number.

The -c option tells grotty to use an output format compatible with paper terminals, like the Teletype machines for which roff and nroff were first developed but which are no longer in wide use. SGR escape sequences (from ISO 6429) are not emitted. Instead,grotty overstrikes, representing a bold character *c* with the sequence '*c* BACKSPACE *c*' and an italic character *c* with the sequence '_ BACKSPACE *c*'. Furthermore, color output is disabled. The same effect can be achieved either by setting the GROFF_NO_SGR environment variable or by using a groff escape sequence within the document; see the subsection "Device control commands" of the *grotty*(1) man page for details.

The legacy output format can be rendered on a video terminal (or emulator) by piping grotty's output through ul, which may render bold italics as reverse video. Some implementations of more are also able to display these sequences; you may wish to experiment

with that command's `-b` option. `less` renders legacy bold and italics without requiring options. In contrast to the teletype output drivers of some other `roff` implementations, `grotty` never outputs reverse line feeds. There is therefore no need to filter its output through `col`.

7.3. `grops`

The postprocessor `grops` translates the output from GNU `troff` into a form suitable for Adobe PostScript devices. It is fully documented on its man page, *grops*(1).

7.3.1. Invoking `grops`

The postprocessor `grops` accepts the following command-line options:

`-b`*flags* Use backward compatibility settings given by *flags* as documented in the *grops*(1) man page. Overrides the command `broken` in the `DESC` file.

`-c`*n* Print *n* copies of each page.

`-F`*dir* Put the directory *dir*`/dev`*name* in front of the search path for the font, prologue and device description files, given the target device *name*, usually **ps**.

`-g` Tell the printer to guess the page length. Useful for printing vertically centered pages when the paper dimensions are determined at print time.

`-I`*path ...*
Consider the directory *path* for searching included files specified with relative paths. The current directory is searched as fallback.

`-l` Use landscape orientation.

`-m` Use manual feed.

`-p`*papersize*
Set the page dimensions. Overrides the commands `papersize`, `paperlength`, and `paperwidth` in the `DESC` file. See the*g roff_font*(5) man page for details.

`-P`*prologue*
Use the *prologue* in the font path as the prologue instead of the default `prologue`. Overrides the environment variable `GROPS_PROLOGUE`.

`-w`*n* Set the line thickness to*n*/1000 em. Ov errides the default value *n* = 40.

`-v` Print the version number.

7.3.2. Embedding PostScript

The escape sequence

'`\X'ps: import` *file llx lly urx ury width* [*height*]''

places a rectangle of the specified*width* containing the PostScript drawing from file*file* bound by the box from*llx lly* to *urx ury* (in PostScript coordinates) at the insertion point. If *height* is not specified, the embedded drawing is scaled proportionally.

See Miscellaneous, for the `psbb` request, which automatically generates the bounding box.

This escape sequence is used internally by the macro `PSPIC` (see the *groff_tmac*(5) man page).

7.4. gropdf

The postprocessor `gropdf` translates the output from GNU `troff` into a form suitable for Adobe PDF devices. It is fully documented on its man page, *gropdf*(1).

7.4.1. Invoking gropdf

The postprocessor `gropdf` accepts the following command-line options:

`-d` Produce uncompressed PDFs that include debugging comments.

`-e` This forces `gropdf` to embed all used fonts in the PDF, even if they are one of the 14 base Adobe fonts.

`-F`*dir* Put the directory *dir*`/devname` in front of the search path for the font, prologue and device description files, given the target device *name*, usually **pdf**.

`-y`*foundry*
This forces the use of a different font foundry.

`-l` Use landscape orientation.

`-p`*papersize*
Set the page dimensions. Overrides the commands `papersize`, `paperlength`, and `paperwidth` in the `DESC` file. See the*g roff_font*(5) man page for details.

`-v` Print the version number.

`-s` Append a comment line to end of PDF showing statistics, i.e. number of pages in document. Ghostscript's *ps2pdf*(1) complains about this line if it is included, but works anyway.

`-u`*filename*
`gropdf` normally includes a ToUnicode CMap with any font created using `text.enc` as the encoding file, this makes it easier to search for words that contain ligatures. You can include your own CMap by specifying a*filename* or ha ve no CMap at all by omitting the*filename*.

7.4.2. Embedding PDF

The escape sequence

'`\X'pdf: pdfpic` *file alignment width* [*height*] [*linelength*]''

places a rectangle of the specified*width* containing the PDF dr awing from file*file* of desired *width* and *height* (if *height* is missing or zero then it is scaled proportionally). If*alignment* is `-L` the drawing is left aligned. If it is `-C` or `-R` a *linelength* greater than the width of the drawing is required as well. If*width* is specified as z ero then the width is scaled in proportion to the height.

7.5. grodvi

The postprocessor `grodvi` translates the output from GNU `troff` into the DVI format produced by the TEX document preparation system. This enables the use of programs that process the DVI format, like `dvips` and `dvipdf`, with GNU `troff` output. `grodvi` is fully documented in its man page, *grodvi*(1).

7.5.1. Invoking grodvi

The postprocessor `grodvi` accepts the following command-line options:

`-d` Do not use **tpic** specials to implement drawing commands.

`-F`*dir* Put the directory *dir*`/devname` in front of the search path for the font and device description files, given the target device *name*, usually **dvi**.

`-l` Use landscape orientation.

`-p`*papersize*
Set the page dimensions. Overrides the commands `papersize`, `paperlength`, and `paperwidth` in the `DESC` file. See the*g roff_font*(5) man page for details.

`-v` Print the version number.

`-w`*n* Set the line thickness to*n*/1000 em. Ov errides the default value *n* = 40.

7.6. grolj4

The postprocessor `grolj4` translates the output from GNU `troff` into the **PCL5** output format suitable for printing on a **HP LaserJet 4** printer. It is fully documented on its man page, *grolj4*(1).

7.6.1. Invoking grolj4

The postprocessor `grolj4` accepts the following command-line options:

`-c`*n* Print *n* copies of each page.

`-F`*dir* Put the directory *dir*`/devname` in front of the search path for the font and device description files, given the target device *name*, usually **lj4**.

`-l` Use landscape orientation.

`-p`*size* Set the page dimensions. Valid values for *size* are: `letter`, `legal`, `executive`, `a4`, `com10`, `monarch`, `c5`, `b5`, `d1`.

`-v` Print the version number.

`-w`*n* Set the line thickness to*n*/1000 em. Ov errides the default value *n* = 40.

The special drawing command '`\D'R` *dh dv*`'` draws a horizontal rectangle from the current position to the position at offset (*dh*,*dv*).

7.7. grolbp

The postprocessor grolbp translates the output from GNU troff into the **LBP** output format suitable for printing on **Canon CAPSL** printers. It is fully documented on its man page, *grolbp*(1).

7.7.1. Invoking grolbp

The postprocessor grolbp accepts the following command-line options:

-c*n* Print *n* copies of each page.

-F*dir* Put the directory *dir*/dev*name* in front of the search path for the font, prologue and device description files, given the target device *name*, usually **lbp**.

-l Use landscape orientation.

-o*orientation*
Use the *orientation* specified: portrait or landscape.

-p*papersize*
Set the page dimensions. See*g roff_font*(5) man page for details.

-w*n* Set the line thickness to*n*/1000 em. Ov errides the default value *n* = 40.

-v Print the version number.

-h Print command-line help.

7.8. grohtml

The grohtml front end (which consists of a preprocessor, pre-grohtml, and a device driver, post-grohtml) translates the output of GNU troff to HTML. Users should always invoke grohtml via the groff command with a \-Thtml option. If no files are given, grohtml will read the standard input. A filename of - will also cause grohtml to read the standard input. HTML output is written to the standard output. When grohtml is run by groff, options can be passed to grohtml using groff's -P option.

grohtml invokes groff twice. In the first pass, pictures, equations, and tables are rendered using the ps device, and in the second pass HTML output is generated by the html device.

grohtml always writes output in UTF-8 encoding and has built-in entities for all non-composite Unicode characters. In spite of this, groff may issue warnings about unknown special characters if they can't be found during the first pass. Such warnings can be safely ignored unless the special characters appear inside a table or equation, in which case glyphs for these characters must be defined for the ps device as well.

This output device is fully documented on its man page, *grohtml*(1).

7.8.1. Invoking grohtml

The postprocessor `grohtml` accepts the following command-line options:

`-a`*bits* Use this number of*bits* (= 1, 2 or 4) f or text antialiasing. Default: *bits* = 4.

`-a0` Do not use text antialiasing.

`-b` Use white background.

`-D`*dir* Store rendered images in the directory *dir*.

`-F`*dir* Put the directory *dir*`/dev`*name* in front of the search path for the font, prologue and device description files, given the target device *name*, usually **html**.

`-g`*bits* Use this number of*bits* (= 1, 2 or 4) f or antialiasing of drawings. Default: *bits* = 4.

`-g0` Do not use antialiasing for drawings.

`-h` Use the `B` element for section headings.

`-i`*resolution*
Use the *resolution* for rendered images. Default: *resolution* = 100 dpi.

`-I`*stem* Set the images'*stem name*. Def ault: *stem* = `grohtml-`*XXX* (*XXX* is the process ID).

`-j`*stem* Place each section in a separate file called *stem*`-`*n*`.html` (where *n* is a generated section number).

`-l` Do not generate the table of contents.

`-n` Generate simple fragment identifiers.

`-o`*offset* Use vertical padding*offset* f or images.

`-p` Display the page rendering progress to `stderr`.

`-r` Do not use horizontal rules to separate headers and footers.

`-s`*size* Set the base font size, to be modified using the elements `BIG` and `SMALL`.

`-S`*level* Generate separate files for sections at level *level*.

`-v` Print the version number.

`-V` Generate a validator button at the bottom.

`-y` Generate a signature of `groff` after the validator button, if any.

7.8.2. `grohtml` specific registers and strings

`\n[ps4html]`
`\*[www-image-template]`
The registers `ps4html` and `www-image-template` are defined by the `pre-grohtml` preprocessor. `pre-grohtml` reads in the `troff` input, marks up the inline equations and passes the result firstly to

```
troff -Tps -rps4html=1 -dwww-image-template=template
```

and secondly to

```
troff -Thtml
```

or

```
troff -Txhtml
```

The PostScript device is used to create all the image files (for `-Thtml`; if `-Txhtml` is used, all equations are passed to `geqn` to produce MathML, and the register `ps4html` enables the macro sets to ignore floating keeps, footers, and headings.

The register `www-image-template` is set to the user specified template name or the default name.

7.9. `gxditview`

7.9.1. Invoking `gxditview`

8. File formats

All files read and written by `gtroff` are text files. The following two sections describe their format.

8.1. `gtroff` Output

This section describes the `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 b y 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.[67]

The main purpose of the intermediate output concept is to facilitate the development of postprocessors by providing a common programming interface for all devices. It has a language of its own that is completely different from the `gtroff` language. While the `gtroff` language is a high-level programming language for text processing, the intermediate output language is a kind of low-level assembler language by specifying all positions on the page for writing and drawing.

The intermediate output produced by `gtroff` is fairly readable, while output from AT&T `troff` is rather hard to understand because of strange habits that are still supported, but not used any longer by `gtroff`.

8.1.1. Language Concepts

During the run of `gtroff`, the input data is cracked down to the information on what has to be printed at what position on the intended device. So the language of the intermediate output format can be quite small. Its only elements are commands with and without arguments. In this section, the term *command* alw ays refers to the intermediate output language, and never to the `gtroff` language used for document formatting. There are commands for positioning and text writing, for drawing, and for device controlling.

8.1.1.1. Separation

AT&T `troff` output has strange requirements 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

[67] The parser and postprocessor for intermediate output can be found in the file *groff-source-dir*`/src/libs/libdriver/input.cpp`.

happens when variable-length command names, arguments, argument lists, or command clusters meet. Commands and arguments with a known, fixed length need not be separated by syntactical space.

A line break is a syntactical element, too. Every command argument can be followed by whitespace, a comment, or a newline character. Thus a*syntactical line break* is defined to consist of optional syntactical space that is optionally followed by a comment, and a newline character.

The normal commands, those for positioning and text, consist of a single letter taking a fixed number of arguments. For historical reasons, the parser allows stacking of such commands on the same line, but fortunately, in `gtroff`'s intermediate output, every command with at least one argument is followed by a line break, thus providing excellent readability.

The other commands—those for drawing and device controlling—have a more complicated structure; some recognize long command names, and some take a variable number of arguments. So all '`D`' and '`x`' commands were designed to request a syntactical line break after their last argument. Only one command, '`x X`', has an argument that can stretch over several lines; all other commands must have all of their arguments on the same line as the command, i.e., the arguments may not be split by a line break.

Empty lines (these are lines containing only space and/or a comment), can occur everywhere. They are just ignored.

8.1.1.2. Argument Units

Some commands take integer arguments that are assumed to represent values in a measurement unit, but the letter for the corresponding scale indicator is not written with the output command arguments. Most commands assume the scale indicator '`u`', the basic unit of the device, some use '`z`', the scaled point unit of the device, while others, such as the color commands, expect plain integers.

Single characters can have the eighth bit set, as can the names of fonts and special characters. The names of characters and fonts can be of arbitrary length. A character that is to be printed is always in the current font.

A string argument is always terminated by the next whitespace character (space, tab, or newline); an embedded '`#`' character is regarded as part of the argument, not as the beginning of a comment command. An integer argument is already terminated by the next non-digit character, which then is regarded as the first character of the next argument or command.

8.1.1.3. Document Parts

A correct intermediate output document consists of two parts, the *prologue* and the *body*.

The task of the prologue is to set the general device parameters using three exactly specified commands. `gtroff`'s prologue is guaranteed to consist of the following three lines (in that order):

```
x T device
x res n h v
x init
```

with the arguments set as outlined in Device Control Commands. 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.

8.1.2. Command Reference

This section describes all intermediate output commands, both from AT&T `troff` as well as the `gtroff` extensions.

8.1.2.1. Comment Command

`#`*anything*<end of line>
: A comment. Ignore any characters from the '`#`' character up to the next newline character.

  This command is the only possibility for commenting in the intermediate output. Each comment can be preceded by arbitrary syntactical space; every command can be terminated by a comment.

8.1.2.2. Simple Commands

The commands in this subsection have a command code consisting of a single character, taking a fixed number of arguments. Most of them are commands for positioning and text writing. These commands are tolerant of whitespace. Optionally, syntactical space can be inserted before, after, and between the command letter and its arguments. All of these commands are stackable; i.e., they can be preceded by other simple commands or followed by arbitrary other commands on the same line. A separating syntactical space is only necessary when two integer arguments would clash or if the preceding argument ends with a string argument.

`C` *xxx*<whitespace>
: Print a special character named *xxx*. The trailing syntactical space or line break is necessary to allow glyph names of arbitrary length. The glyph is printed at the current print position; the glyph's size is read from the font file. The print position is not changed.

`c` *g*
: Print glyph *g* at the current print position;[68] the glyph's size is read from the font file. The print position is not changed.

[68] '`c`' is actually a misnomer since it outputs a glyph.

`f` *n* Set font to font number *n* (a non-negative integer).

`H` *n* Move right to the absolute vertical position*n* (a non-negativ e 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 65536. The number of color components and their meaning vary for the different color schemes. These commands are generated by `gtroff`'s escape sequence `\m`. No position changing. These commands are a `gtroff` extension.

`mc` *cyan magenta yellow*
Set color using the CMY color scheme, having the 3 color components *cyan*, *magenta*, and *yellow*.

`md` Set color to the default color value (black in most cases). No component arguments.

`mg` *gray* Set color to the shade of gray given by the argument, an integer between 0 (black) and 65536 (white).

`mk` *cyan magenta yellow black*
Set color using the CMYK color scheme, having the 4 color components *cyan*, *magenta*, *yellow*, and *black*.

`mr` *red green blue*
Set color using the RGB color scheme, having the 3 color components *red*, *green*, and *blue*.

`N` *n* Print glyph with index *n* (a non-negative integer) of the current font. This command is a `gtroff` extension.

`n` *b a* Inform the device about a line break, but no positioning is done by this command. In AT&T `troff`, the integer arguments*b* and *a* inf ormed 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 point size to *n* scaled points (this is unit 'z'). A T&T `troff` used the unit points ('p') instead. See Output Language Compatibility.

`t` *xxx*<whitespace>
`t` *xxx dummy-arg*<whitespace>
: Print a word, i.e., a sequence of characters *xxx* representing output glyphs which names are single characters, terminated by a space character or a line break; an optional second integer argument is ignored (this allows the formatter to generate an even number of arguments). The first glyph should be printed at the current position, the current horizontal position should then be increased by the width of the first glyph, and so on for each glyph. The widths of the glyphs are read from the font file, scaled for the current point size, and rounded to a multiple of the horizontal resolution. Special characters cannot be printed using this command (use the 'C' command for special characters). This command is a `gtroff` extension; it is only used for devices whose `DESC` file contains the `tcommand` keyword (see `DESC` File Format).

`u` *n xxx*<whitespace>
: Print word with track kerning. This is the same as the 't' command except that after printing each glyph, the current horizontal position is increased by the sum of the width of that glyph and *n* (an integer in basic units 'u'). This command is a `gtroff` extension; it is only used for devices whose `DESC` file contains the `tcommand` keyword (see `DESC` File Format).

`V` *n*
: Move down to the absolute vertical position *n* (a non-negativ e 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.

8.1.2.3. Graphics Commands

Each graphics or drawing command in the intermediate output starts with the letter 'D', followed by one or two characters that specify a subcommand; this is followed by a fixed or variable number of integer arguments that are separated by a single space character. A 'D' command may not be followed by another command on the same line (apart from a comment), so each 'D' command is terminated by a syntactical line break.

`gtroff` output follows the classical spacing rules (no space between command and subcommand, all arguments are preceded by a single space character), but the parser allows optional space between the command letters and makes the space before the first argument optional. As usual, each space can be any sequence of tab and space characters.

Some graphics commands can take a variable number of arguments. In this case, they are integers representing a size measured in basic units 'u'. The arguments called *h1*, *h2*, ..., *hn* stand for horizontal distances where positive means right, negative left. The arguments called *v1*, *v2*, ..., *vn* stand f or vertical distances where positive means down, negative up. All these distances are offsets relative to the current location.

Each graphics command directly corresponds to a similar `gtroff \D` escape sequence. See Drawing Requests.

Unknown '`D`' commands are assumed to be device-specific. Its arguments are parsed as strings; the whole information is then sent to the postprocessor.

In the following command reference, the syntax element <line break> means a syntactical line break as defined above.

`D~` *h1 v1 h2 v2 ... hn vn*<line break>
: Draw B-spline from current position to offset (*h1*,*v1*), then to offset (*h2*,*v2*), if given, etc. up to (*hn*,*vn*). This command takes a variable number of argument pairs; the current position is moved to the terminal point of the drawn curve.

`Da` *h1 v1 h2 v2*<line break>
: Draw arc from current position to (*h1*,*v1*)+(*h2*,*v2*) with center at (*h1*,*v1*); then move the current position to the final point of the arc.

`DC` *d*<line break>
`DC` *d dummy-arg*<line break>
: Draw a solid circle using the current fill color with diameter*d* (integer in basic units '`u`') with leftmost point at the current position; then move the current position to the rightmost point of the circle. An optional second integer argument is ignored (this allows the formatter to generate an even number of arguments). This command is a `gtroff` extension.

`Dc` *d*<line break>
: Draw circle line with diameter*d* (integer in basic units '`u`') with leftmost point at the current position; then move the current position to the rightmost point of the circle.

`DE` *h v*<line break>
: Draw a solid ellipse in the current fill color with a horizontal diameter of*h* and a vertical diameter of*v* (both integers in basic units '`u`') with the leftmost point at the current position; then move to the rightmost point of the ellipse. This command is a `gtroff` extension.

`De` *h v*<line break>
: Draw an outlined ellipse with a horizontal diameter of*h* and a v ertical diameter of *v* (both integers in basic units '`u`') with the leftmost point at current position; then move to the rightmost point of the ellipse.

`DF` *color-scheme* `[`*component ...*`]`<line break>
: Set fill color for solid drawing objects using different color schemes; the analogous command for setting the color of text, line graphics, and the outline of graphic objects is '`m`'. The color components are specified as integer arguments between 0 and 65536. The number of color components and their meaning vary for the different color schemes. These commands are generated by `gtroff`'s escape sequences '`\D'F ...''` and`\M` (with no other corresponding graphics commands). No position changing. This command is a `gtroff` extension.

`DFc` *cyan magenta yellow*<line break>
: Set fill color for solid drawing objects using the CMY color scheme, having the 3 color components*cy an*, *magenta*, and *yellow*.

`DFd`<line break>
: Set fill color for solid drawing objects to the default fill color value (black in most cases). No component arguments.

`DFg` *gray*<line break>
: Set fill color for solid drawing objects to the shade of gray given by the argument, an integer between 0 (black) and 65536 (white).

`DFk` *cyan magenta yellow black*<line break>
: Set fill color for solid drawing objects using the CMYK color scheme, having the 4 color components*cy an*, *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*, *g reen*, and *blue*.

`Df` *n*<line break>
: The argument *n* must be an integer in the range -32767 to 32767.

$0 \leq n \leq 1000$
: Set the color for filling solid drawing objects to a shade of gray, where 0 corresponds to solid white, 1000 (the default) to solid black, and values in between to intermediate shades of gray; this is obsoleted by command '`DFg`'.

$n < 0$ or $n > 1000$
: Set the filling color to the color that is currently being used for the text and the outline, see command '`m`'. For example, the command sequence

```
mg 0 0 65536
Df -1
```

: sets all colors to blue.

No position changing. This command is a `gtroff` extension.

`Dl` *h v*<line break>
: Draw line from current position to offset (*h*,*v*) (integers in basic units '`u`'); then set current position to the end of the drawn line.

`Dp` *h1 v1 h2 v2 ... hn vn*<line break>
: Draw a polygon line from current position to offset (*h1*,*v1*), from there to offset (*h2*,*v2*), etc. up to offset (*hn*,*vn*), and from there back to the starting position. For historical reasons, the position is changed by adding the sum of all arguments with odd index to the actual horizontal position and the even ones to the vertical position. Although this doesn't make sense it is kept for compatibility. This command is a `gtroff` extension.

`DP` *h1 v1 h2 v2 ... hn vn*<line break>
: Draw a solid polygon in the current fill color rather than an outlined polygon, using the same arguments and positioning as the corresponding '`Dp`' command. This command is a `gtroff` extension.

`Dt` *n*<line break>
: Set the current line thickness to *n* (an integer in basic units '`u`') if *n*>0; if *n*=0 select the smallest available line thickness; if *n*<0 set the line thic kness proportional to the point size (this is the default before the first '`Dt`' command was specified). For historical reasons, the horizontal position is changed by adding the argument to the actual horizontal position, while the vertical position is not changed. Although this doesn't make sense it is kept for compatibility. This command is a `gtroff` extension.

8.1.2.4. Device Control Commands

Each device control command starts with the letter '`x`', followed by a space character (optional or arbitrary space or tab in `gtroff`) and a subcommand letter or word; each argument (if any) must be preceded by a syntactical space. All '`x`' commands are terminated by a syntactical line break; no device control command can be followed by another command on the same line (except a comment).

The subcommand is basically a single letter, but to increase readability, it can be written as a word, i.e., an arbitrary sequence of characters terminated by the next tab, space, or newline character. All characters of the subcommand word but the first are simply ignored. For example, `gtroff` outputs the initialization command '`x i`' as '`x init`' and the resolution command '`x r`' as '`x res`'.

In the following, the syntax element <line break> means a syntactical line break (see Separation).

`xF` *name*<line break>
: The '`F`' stands for *Filename*.

  Use *name* as the intended name for the current file in error reports. This is useful for remembering the original file name when `gtroff` uses an internal piping mechanism. The input file is not changed by this command. This command is a `gtroff` extension.

`xf` *n s*<line break>
: The '`f`' stands for *font*.

  Mount font position *n* (a non-negativ e integer) with font named *s* (a te xt word). See Font Positions.

`xH` *n*<line break>
: The '`H`' stands for *Height*.

  Set glyph height to *n* (a positiv e integer in scaled points '`z`'). AT&T `troff` uses the unit points ('`p`') instead. See Output Language Compatibility.

`xi`<line break>
: The '`i`' stands for *init*.

  Initialize device. This is the third command of the prologue.

`xp`<line break>
: The '`p`' stands for *pause*.

  Parsed but ignored. The 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.

8.1.2.5. Obsolete Command

In AT&T `troff` output, the writing of a single glyph is mostly done by a very strange command that combines a horizontal move and a single character giving the glyph name. It doesn't have a command code, but is represented by a 3-character argument consisting of exactly 2 digits and a character.

dd g Move right*dd* (e xactly two decimal digits) basic units '`u`', then print glyph *g* (represented as a single character).

In GNU `troff`, arbitrary syntactical space around and within this command is allowed. Only when a preceding command on the same line ends with an argument of variable length is a separating space obligatory. In AT&T `troff`, large clusters of these and other commands are used, mostly without spaces; this made such output almost unreadable.

For modern high-resolution devices, this command does not make sense because the width of the glyphs can become much larger than two decimal digits. In`gtroff`, this is only used for the devices `X75`, `X75-12`, `X100`, and `X100-12`. For other devices, the commands '`t`' and '`u`' provide a better functionality.

8.1.3. Intermediate Output Examples

This section presents the intermediate output generated from the same input for three different devices. The input is the sentence '`hell world`' fed into `gtroff` on the command line.

High-resolution device `ps`

This is the standard output of `gtroff` if no `-T` option is given.

```
shell> echo "hell world" | groff -Z -T ps

x T ps
x res 72000 1 1
x init
p1
x font 5 TR
f5
s10000
V12000
H72000
thell
wh2500
tw
H96620
torld
n12000 0
x trailer
V792000
x stop
```

This output can be fed into `grops` to get its representation as a PostScript file.

Low-resolution device `latin1`

This is similar to the high-resolution device except that the positioning is done at a minor scale. Some comments (lines starting with '#') were added for clarification; they were not generated by the formatter.

```
shell> echo "hell world" | groff -Z -T latin1

# prologue
x T latin1
x res 240 24 40
x init
# begin a new page
p1
# font setup
x font 1 R
f1
s10
# initial positioning on the page
V40
H0
# write text 'hell'
thell
# inform about space, and issue a horizontal jump
wh24
# write text 'world'
tworld
# announce line break, but do nothing because...
n40 0
# ...the end of the document has been reached
x trailer
V2640
x stop
```

This output can be fed into `grotty` to get a formatted text document.

AT&T `troff` output

Since a computer monitor has a much lower resolution than modern printers, the intermediate output for X11 devices can use the jump-and-write command with its 2-digit displacements.

```
shell> echo "hell world" | groff -Z -T X100

x T X100
x res 100 1 1
x init
p1
x font 5 TR
f5
s10
V16
H100
```

```
# write text with jump-and-write commands
ch07e07l03lw06w11o07r05l03dh7
n16 0
x trailer
V1100
x stop
```

This output can be fed into `xditview` or `gxditview` for displaying in X.

Due to the obsolete jump-and-write command, the text clusters in the AT&T `troff` output are almost unreadable.

8.1.4. Output Language Compatibility

The intermediate output language of AT&T `troff` was first documented in *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.

8.2. Device and Font Files

The GNU `troff` font format is a rough superset of the AT&T device-independent `troff` font format. In distinction to the AT&T implementation, GNU `troff` lacks a binary format; all files are text files.[69] The font files for device *name* are stored in a directory `devname`. There are two types of file: a device description file called `DESC` and for each font *f* a font file called *f*.

[69] Plan 9 `troff` has also abandoned the binary format.

8.2.1. DESC File Format

The DESC file can contain the following types of line. Except for the `charset` keyword, which must come last (if at all), the order of the lines is not important. Later entries in the file, however, override previous values.

`charset` This line and everything following in the file are ignored. It is allowed for the sake of backwards compatibility.

`family` *fam*
The default font family is *fam*.

`fonts` *n F1 F2 F3 ... Fn*
Fonts *F1* ... *Fn* are mounted in the font positions*m*+1, ...,*m*+ *n* where *m* is the number of styles. This command may extend over more than one line. A font name of 0 means no font is mounted on the corresponding font position.

`hor` *n* The horizontal resolution is*n* basic units . All horizontal quantities are rounded to be multiples of this value.

`image_generator` *string*
Needed for `grohtml` only. It specifies the program to generate PNG images from PostScript input. Under GNU/Linux this is usually `gs` but under other systems (notably cygwin) it might be set to another name.

`paperlength` *n*
The physical vertical dimension of the output medium in basic units. This isn't used by `troff` itself but by output devices. Deprecated. Use `papersize` instead.

`papersize` *string ...*
Select a paper size. Valid values for *string* are the ISO paper types `A0`–`A7`, `B0`–`B7`, `C0`–`C7`, `D0`–`D7`, `DL`, and the US paper types `letter`, `legal`, `tabloid`, `ledger`, `statement`, `executive`, `com10`, and `monarch`. Case is not significant for *string* if it holds predefined paper types. Alternatively, *string* can be a file name (e.g. `/etc/papersize`); if the file can be opened, `groff` reads the first line and tests for the above paper sizes. Finally, *string* can be a custom paper size in the format *length*`,`*width* (no spaces before and after the comma). Both*length* and *width* must have a unit appended; valid values are '`i`' for inches, '`c`' for centimeters, '`p`' for points, and '`P`' for picas. Example:`12c,235p`. An argument that starts with a digit is always treated as a custom paper format. `papersize` sets both the vertical and horizontal dimension of the output medium.

More than one argument can be specified; `groff` scans from left to right and uses the first valid paper specification.

`paperwidth` *n*
The physical horizontal dimension of the output medium in basic units. This isn't used by `troff` itself but by output devices. Deprecated. Use `papersize` instead.

`pass_filenames`
: Tell `gtroff` to emit the name of the source file currently being processed. This is achieved by the intermediate output command '`F`'. Currently, this is only used by the `grohtml` output device.

`postpro` *program*
: Call *program* as a postprocessor. For example, the line

```
postpro grodvi
```

in the file `devdvi/DESC` makes `groff` call `grodvi` if option `-Tdvi` is given (and `-Z` isn't used).

`prepro` *program*
: Call *program* as a preprocessor. Currently, this keyword is used by `groff` with option `-Thtml` or `-Txhtml` only.

`print` *program*
: Use *program* as a spooler program for printing. If omitted, the `-l` and `-L` options of `groff` are ignored.

`res` *n*
: There are*n* basic units per inch.

`sizes` *s1 s2 ... sn* `0`
: This means that the device has fonts at*s1*, *s2*, ...*sn* scaled points . The list of sizes must be terminated by 0 (this is digit zero). Each*si* can also be a r ange of sizes *m–n*. The list can extend over more than one line.

`sizescale` *n*
: The scale factor for point sizes. By default this has a value of 1. One scaled point is equal to one point/*n*. The arguments to the`unitwidth` and `sizes` commands are given in scaled points. See Fractional Type Sizes.

`styles` *S1 S2 ... Sm*
: The first *m* font positions are associated with styles*S1* ... *Sm*.

`tcommand` This means that the postprocessor can handle the '`t`' and '`u`' intermediate output commands.

`unicode` Indicate that the output device supports the complete Unicode repertoire. Useful only for devices that produce*char acter entities* instead of glyphs.

If `unicode` is present, no `charset` section is required in the font description files since the Unicode handling built into `groff` is used. However, if there are entries in a `charset` section, they either override the default mappings for those particular characters or add new mappings (normally for composite characters).

This is used for `-Tutf8`, `-Thtml`, and `-Txhtml`.

`unitwidth` *n*
: Quantities in the font files are given in basic units for fonts whose point size is *n* scaled points.

`unscaled_charwidths`
: Make the font handling module always return unscaled character widths. Needed for the `grohtml` device.

`use_charnames_in_special`
: This command indicates that `gtroff` should encode special characters inside special commands. Currently, this is only used by the `grohtml` output device. See Postprocessor Access.

`vert` *n*
: The vertical resolution is *n* basic units . All vertical quantities are rounded to be multiples of this value.

The `res`, `unitwidth`, `fonts`, and `sizes` lines are mandatory. Other commands are ignored by `gtroff` but may be used by postprocessors to store arbitrary information about the device in the `DESC` file.

GNU `troff` recognizes but completely ignores the obsolete keywords `spare1`, `spare2`, and `biggestfont`.

8.2.2. Font File Format

A *font file*, also (and probably better) called a *font description file*, has two sections. The first section is a sequence of lines each containing a sequence of blank-delimited words; the first word in the line is a key, and subsequent words give a value for that key.

`name` *f*
: The name of the font is *f*.

`spacewidth` *n*
: The normal width of a space is *n*.

`slant` *n*
: The glyphs of the font have a slant of *n* deg rees. (Positive means forward.)

`ligatures` *lig1 lig2 ... lign* `[0]`
: Glyphs *lig1*, *lig2*, ...,*lign* are ligatures; possib le ligatures are '`ff`', '`fi`', '`fl`', '`ffi`' and '`ffl`'. For backwards compatibility, the list of ligatures may be terminated with a 0. The list of ligatures may not extend over more than one line.

`special`
: The font is *special*; this means that when a glyph is requested that is not present in the current font, it is searched for in any special fonts that are mounted.

Other commands are ignored by `gtroff` but may be used by postprocessors to store arbitrary information about the font in the font file.

The first section can contain comments, which start with the '`#`' character and extend to the end of a line.

The second section contains one or two subsections. It must contain a `charset` subsection and it may also contain a `kernpairs` subsection. These subsections can appear in any order. Each subsection starts with a word on a line by itself.

The word `charset` starts the character set subsection.[70] The `charset` line is followed by a sequence of lines. Each line gives information for one glyph. A line comprises a number of fields separated by blanks or tabs. The format is

name metrics type code [*entity-name*] [– *comment*]

[70] This keyword is misnamed since it starts a list of ordered glyphs, not characters.

name identifies the glyph name:[71] if *name* is a single character *c* then it corresponds to the `gtroff` input character *c*; if it is of the form '`\`*c*' where*c* is a single char acter, then it corresponds to the special character `\[`*c*`]`; otherwise it corresponds to the special character '`\[`*name*`]`'. If it is exactly two characters *xx* it can be entered as '`\(`*xx*'. Single-letter special characters can't be accessed as '`\`*c*'; the only exception is '`\-`', which is identical to `\[-]`.

`gtroff` supports 8-bit input characters; however some utilities have difficulties with eight-bit characters. For this reason, there is a convention that the entity name '`charn`' is equivalent to the single input character whose code is*n*. F or example, '`char163`' would be equivalent to the character with code 163, which is the pounds sterling sign in the ISO Latin-1 character set. You shouldn't use '`charn`' entities in font description files since they are related to input, not output. Otherwise, you get hard-coded connections between input and output encoding, which prevents use of different (input) character sets.

The name '—' is special and indicates that the glyph is unnamed; such glyphs can only be used by means of the `\N` escape sequence in `gtroff`.

The *type* field gives the glyph type:

`1` the glyph has a descender, for example, '`p`';

`2` the glyph has an ascender, for example, '`b`';

`3` the glyph has both an ascender and a descender, for example, '`(`'.

The *code* field gives the code that the postprocessor uses to print the glyph. The glyph can also be input to `gtroff` using this code by means of the `\N` escape sequence. *code* can be any integer. If it starts with '`0`' it is interpreted as octal; if it starts with '`0x`' or '`0X`' it is interpreted as hexadecimal. Note, however, that the `\N` escape sequence only accepts a decimal integer.

The *entity-name* field gives an ASCII string identifying the glyph that the postprocessor uses to print the `gtroff` glyph *name*. This field is optional and has been introduced so that the `grohtml` device driver can encode its character set. For example, the glyph '`\[Po]`' is represented as '`&pound;`' in HTML 4.0.

Anything on the line after the*entity-name* field resp . after '–' is ignored.

The *metrics* field has the form:

width[,*height*[,*depth*[,*italic-correction* [,*left-italic-correction*[,*subscript-correction*]]]]]

There must not be any spaces between these subfields (it has been split here into two lines for better legibility only). Missing subfields are assumed to be 0. The subfields are all decimal integers. Since there is no associated binary format, these values are not required to fit into a variable of type '`char`' as they are in AT&T device-independent `troff`. The *width* subfield gives the width of the glyph. The*height* subfield giv es the height of the glyph (upwards is positive); if a glyph does not extend above the baseline, it should be given a zero height, rather than a negative height. The*depth* subfield giv es the depth of the glyph, that is, the distance from the baseline to the lowest point below the baseline to which the glyph extends (downwards is positive); if a glyph does not extend below the baseline, it should be given a zero depth, rather than a negative depth. The*italic-*

[71] The distinction between input, characters, and output, glyphs, is not clearly separated in the terminology of `groff`; for example, the `char` request should be called `glyph` since it defines an output entity.

correction subfield gives the amount of space that should be added after the glyph when it is immediately to be followed by a glyph from a roman font. The*left-italic-correction* subfield gives the amount of space that should be added before the glyph when it is immediately to be preceded by a glyph from a roman font. The*subscript-correction* giv es the amount of space that should be added after a glyph before adding a subscript. This should be less than the italic correction.

A line in the `charset` section can also have the format

name "

This indicates that*name* is just another name f or the glyph mentioned in the preceding line.

The word `kernpairs` starts the kernpairs section. This contains a sequence of lines of the form:

c1 c2 n

This means that when glyph*c1* appears ne xt to glyph*c2* the space betw een them should be increased by *n*. Most entries in the kernpairs section have a negative value for *n*.

9. Installation

10. Copying This Manual

Version 1.3, 3 November 2008

1 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 e veryone 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.

2 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.

3 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.

4 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.

5 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.

6 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.”

7 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.

8 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.

9 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.

10
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.

11
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.

12
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.


```
@bye=18557
@contents=509
@documentencoding=17
@documentlanguage=16
@end=507, 514, 541, 10552, 10634, 10660, 10705
@finalout=22
@footnotestyle=13
@ifnottex=511, 539
@iftex=10624, 10650, 10695
@noindent=1008, 1521, 1578, 1641, 1650, 1662, 1677, 1713, 1723,
1778, 1785, 1806, 1883, 1902, 1964, 2193, 3206, 3246, 3542, 3557,
4774, 4850, 4876, 4917, 5007, 5337, 5452, 5589, 5684, 5857, 5883,
5911, 5919, 5966, 6007, 6189, 6220, 6227, 6250, 6257, 6399, 6540,
6767, 6774, 7008, 7061, 7201, 7209, 7249, 7459, 7466, 7752, 7773,
7857, 7983, 7994, 8019, 8028, 8103, 8110, 8188, 8369, 8408, 8493,
8502, 8538, 8551, 8576, 8663, 8858, 8877, 9213, 9228, 9507, 9530,
9726, 9735, 9765, 9808, 9942, 10072, 10263, 10285, 10374, 11140,
11214, 11234, 11247, 11254, 11271, 11513, 11637, 11663, 11700,
11892, 11912, 11986, 12021, 12061, 12690, 12718, 12768, 13011,
13609, 13665, 13745, 14043, 14238, 14296, 14377, 14458, 14697,
14748, 14862, 14880, 14911, 14920, 14944, 15056, 15140, 15187,
15484, 15491, 15625, 16129, 16220, 16540, 16547, 16760, 17095,
17099, 17332, 17377, 17410, 17610, 17782, 17851, 17880, 17892
@opindex=5364, 5365, 5366, 5367, 5368, 5379, 5380, 5381, 5382,
5383, 5384, 5392, 5404, 5405, 5406, 5440, 5441, 5466, 5467
@setchapternewpage=12
@setfilename=10
```

```
@smallbook=20
@stindex=11073
@tindex=1379, 1384, 1400, 1411, 1417, 1422, 1423, 1432, 1436,
1444, 1489, 1564
@titlepage=496
@top=513
@vskip=505
man:.de FONT=2475
man:FONT=2475, 2475, 2475, 2475, 2475, 2475, 2475
man:SH=2431, 2431, 2431, 2431, 2431, 2431, 2431, 2475, 2475, 2475,
2475, 2475, 2475, 2475, 2475, 2475, 15714, 15714, 15714,
15714, 15714, 15714, 15714, 15714, 15737, 15737, 15737, 15737,
15737, 15737, 15737, 15761, 15761, 15761, 15761, 15761, 15761,
15761, 15761, 15761, 15792, 15792, 15792, 15792, 15792, 15792,
15792, 15792, 15792, 15792, 15792, 15792, 15792, 15829, 15829,
15829, 15829, 15829, 15829, 15829, 15852, 15852, 15852, 15852,
15852, 15852, 15852, 15852, 15852, 15875, 15875, 15875, 15875,
15875, 15897, 15897, 15897, 15897, 15897
man:hw=2475
man:mso=15875
```

B Request Index

C Escape Index

E Register Index

F Macro Index

G String Index

J Program and File Index

K Concept Index

- b -

- c -

- d -

- e -

- h -

- i -

- m -

- q -

- r -

- t -

- u -