groff

The GNU implementation of `troff`
version 1.25.0.rc3.8-b00b9
October 2026

Trent A. Fisher
Werner Lemberg
G. Branden Robinson

This manual documents GNU `troff` version 1.25.0.rc3.8-b00b9.

Copyright © 1994–2026 Free Software Foundation, Inc.
Copyright © 2018–2026 G. Branden Robinson

This document is dual-licensed. You may distribute and/or modify it under the terms of either or both of the following licenses.[1]

- GNU GPLv3:

  > This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version. A copy of the license is included in the section entitled "GNU General Public License".
  >
  > This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.
  >
  > You should have received a copy of the GNU General Public License along with this program. If not, see `http://www.gnu.org/licenses/`.

- GNU FDLv1.3:

  > 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".

[1] We treat this manual as a "program" because its source format, Texinfo, permits embedding of arbitrary TEX code, and TEX, like `troff`, is a Turing-complete language.

Table of Contents

1 Introduction . . . 1

1.1 Background . . . 1
1.2 What Is `groff`? . . . 1
1.3 GNU `troff` Capabilities . . . 2
1.4 Macro Packages . . . 2
1.5 Preprocessors . . . 3
1.6 Output Devices . . . 3
1.7 Installation . . . 3
1.8 Conventions Used in This Manual . . . 4
1.9 Credits . . . 5

2 Invoking `groff` . . . 7

2.1 Options . . . 7
2.2 Environment . . . 13
2.3 Macro Directories . . . 14
2.4 Font Directories . . . 15
2.5 Paper Format . . . 16
2.6 Invocation Examples . . . 16

3 Tutorial for Macro Package Users . . . 19

3.1 Basics . . . 19
3.2 Common Features . . . 21
3.2.1 Paragraphs . . . 21
3.2.2 Sections and Chapters . . . 22
3.2.3 Headers and Footers . . . 22
3.2.4 Page Layout . . . 22
3.2.5 Displays and Keeps . . . 23
3.2.6 Footnotes and Endnotes . . . 23
3.2.7 Table of Contents . . . 23
3.2.8 Indexing . . . 23
3.2.9 Document Formats . . . 24
3.2.10 Columnation . . . 24
3.2.11 Font and Size Changes . . . 24
3.2.12 Predefined Text . . . 24
3.2.13 Preprocessor Support . . . 24
3.2.14 Configuration and Customization . . . 24

4 Macro Packages . . . 25

4.1 `man` . . . 25
4.1.1 Optional `man` extensions . . . 25

Custom headers and footers . . . 25
Ultrix-specific man macros . . . 26
Simple example . . . 27
4.2 `mdoc` . . . 27
4.3 `me` . . . 28
4.4 `mm` . . . 28
4.5 `mom` . . . 28
4.6 `ms` . . . 28
4.6.1 Introduction . . . 29
4.6.1.1 Basic information . . . 29
4.6.2 Document Structure . . . 31
4.6.3 Document Control Settings . . . 32
Margin settings . . . 32
Titles (headers, footers) . . . 33
Text settings . . . 34
Paragraph settings . . . 34
Heading settings . . . 35
Footnote settings . . . 36
Display settings . . . 37
Other settings . . . 37
4.6.4 Document Description Macros . . . 38
4.6.5 Body Text . . . 39
4.6.5.1 Text settings . . . 40
4.6.5.2 Typographical symbols . . . 40
4.6.5.3 Paragraphs . . . 40
4.6.5.4 Headings . . . 42
4.6.5.5 Typeface and decoration . . . 45
4.6.5.6 Lists . . . 47
4.6.5.7 Indented regions . . . 49
4.6.5.8 Keeps, boxed keeps, and displays . . . 50
4.6.5.9 Tables, figures, equations, and references . . . 52
4.6.5.10 Footnotes . . . 54
4.6.5.11 Language and localization . . . 55
4.6.6 Page layout . . . 56
4.6.6.1 Headers and footers . . . 56
4.6.6.2 Tab stops . . . 57
4.6.6.3 Margins . . . 57
4.6.6.4 Multiple columns . . . 57
4.6.6.5 Creating a table of contents . . . 58
4.6.7 Using PDF features with `ms` . . . 60
4.6.8 Differences from AT&T `ms` . . . 61
4.6.8.1 Unix Version 7 `ms` macros unimplemented by `groff ms` . 63
4.6.9 Legacy Features . . . 63
AT&T accent mark strings . . . 63
Berkeley accent mark and glyph strings . . . 64
4.6.10 Naming Conventions . . . 65

5 GNU `troff` Reference 67
5.1 Text 67
5.1.1 Filling 67
5.1.2 Sentences 68
5.1.3 Hyphenation 69
5.1.4 Breaking 70
5.1.5 Adjustment 71
5.1.6 Tabs and Leaders 71
5.1.7 Requests and Macros 71
5.1.8 Macro Packages 74
5.1.9 Input Format 74
5.1.10 Input Encodings 75
5.1.11 Input Conventions 76
5.2 Page Geometry 79
5.3 Measurements 80
5.3.1 Motion Quanta 81
5.3.2 Default Units 81
5.4 Numeric Expressions 82
5.5 Identifiers 86
5.6 Formatter Instructions 89
5.6.1 Control Characters 89
5.6.2 Invoking Requests 90
5.6.3 Calling Macros 92
5.6.4 Using Escape Sequences 93
5.6.5 Delimiters 96
5.7 Comments 97
5.8 Registers 99
5.8.1 Setting Registers 99
5.8.2 Interpolating Registers 102
5.8.3 Auto-increment 102
5.8.4 Assigning Register Formats 103
5.8.5 Built-in Registers 105
5.9 Manipulating Filling and Adjustment 106
5.10 Manipulating Hyphenation 114
5.11 Manipulating Spacing 122
5.12 Tabs and Fields 125
5.12.1 Leaders 127
5.12.2 Fields 128
5.13 Character Translations 129
5.14 `troff` and `nroff` Modes 130
5.15 Line Layout 131
5.16 Line Continuation 134
5.17 Page Layout 136
5.18 Page Control 137
5.19 Using Fonts 140

5.19.1 Selecting Fonts ... 141
5.19.2 Font Families ... 143
5.19.3 Font Mounting Positions ... 145
5.19.4 Characters and Glyphs ... 146
5.19.5 Character Classes ... 154
5.19.6 Special Fonts ... 155
5.19.7 Font Effects and Text Underlining ... 156
5.19.8 Ligatures and Kerning ... 158
5.19.9 Italic Corrections ... 160
5.19.10 Dummy Characters ... 160
5.20 Manipulating Type Size and Vertical Spacing ... 163
5.20.1 Changing the Type Size ... 163
5.20.2 Changing the Vertical Spacing ... 164
5.20.3 Using Fractional Type Sizes ... 166
5.21 Colors ... 168
5.22 Strings ... 170
5.23 Conditionals and Loops ... 175
5.23.1 Operators in Conditionals ... 175
5.23.2 if-then ... 178
5.23.3 if-else ... 178
5.23.4 Conditional Blocks ... 179
5.23.5 while ... 181
5.24 Writing Macros ... 182
5.24.1 Parameters ... 186
5.24.2 Copy Mode ... 188
5.25 Page Motions ... 191
5.26 Output Line Annotation ... 197
5.27 Drawing Geometric Objects ... 200
5.28 Deferring Output ... 204
5.29 Traps ... 206
5.29.1 Vertical Position Traps ... 206
5.29.1.1 Page Location Traps ... 206
5.29.1.2 The Implicit Page Trap ... 211
5.29.1.3 Diversion Traps ... 211
5.29.2 Input Line Traps ... 212
5.29.3 Blank Line Traps ... 214
5.29.4 Leading Space Traps ... 214
5.29.5 End-of-input Traps ... 214
5.30 Diversions ... 217
5.31 Punning Names ... 222
5.32 Environments ... 224
5.33 Suppressing Output ... 227
5.34 Host System Service Access ... 228
5.35 Postprocessor Access ... 235
5.36 Miscellaneous ... 236
5.37 GNU `troff` Internals ... 237

5.38 Debugging ... 240
5.38.1 Warnings ... 245
5.39 Implementation Differences ... 248
5.39.1 Safer Mode ... 248
5.39.2 Compatibility Mode ... 248
5.39.3 Other Differences ... 251

6 File Formats ... 257
6.1 Device and Font Description Files ... 257
6.1.1 `DESC` File Format ... 257
6.1.2 Font Description File Format ... 260
6.2 GNU `troff` Output ... 263
6.2.1 Language Concepts ... 264
6.2.1.1 Syntax ... 264
6.2.1.2 Argument Units ... 265
6.2.1.3 Output Structure ... 265
6.2.2 Command Reference ... 266
6.2.2.1 Comment Command ... 266
6.2.2.2 Simple Commands ... 266
6.2.2.3 Graphics Commands ... 268
6.2.2.4 Device Control Commands ... 271
6.2.2.5 Legacy Compressed Encoding ... 273
6.2.3 GNU `troff` Output Examples ... 274
6.2.4 Output Language Compatibility ... 276

Appendix A Copying This Manual ... 279
A.1 GNU General Public License ... 279
A.2 GNU Free Documentation License ... 292

Appendix B Indices ... 301
B.1 Request Index ... 301
B.2 Escape Sequence Index ... 304
B.3 Operator Index ... 305
B.4 Register Index ... 306
B.5 Macro Index ... 309
B.6 String Index ... 311
B.7 File Keyword Index ... 313
B.8 Program and File Index ... 314
B.9 Concept Index ... 316

1 Introduction

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

1.1 Background

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

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

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

1.2 What Is `groff`?

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

A reimplementation and extension of `troff` and other programs from AT&T Unix, `groff` is widely available on POSIX and other systems owing to its long association with Unix manuals, including man pages. It and its

predecessor have produced several best-selling software engineering texts. `groff` can create typographically sophisticated documents while consuming minimal system resources.

Like its predecessor "troff", the term "groff" affords two popular pronunciations: as one syllable (like the surname), rhyming with "trough", or as "jee-roff", in analogy to the Bell Labs pronunciation "tee-roff". Little risk of confusion exists; use whichever suits you.

The architecture of the GNU `roff` system follows that of other device-independent `roff` implementations, comprising preprocessors, macro packages, output drivers (or "postprocessors"), and a suite of utilities, with the formatter program `troff` at its heart.

The front end programs available in the GNU `roff` system make it easier to use than traditional `roff`s that required the explicit construction of pipelines or use of temporary files to carry a source document from maintainable form to device-ready output.

1.3 GNU `troff` Capabilities

GNU `troff` is a typesetting document formatting program; it provides a wide range of low-level text and page operations within the framework of a programming language. These operations compose to generate footnotes, tables of contents, mathematical equations, diagrams, multi-column text, and other elements of typeset works. Here is a survey of formatter features; all are under precise user control.

- text filling, breaking, alignment to the left or right margin; centering
- adjustment of inter-word space size to justify text, and of inter-sentence space size to suit local style conventions
- automatic and manual determination of hyphenation points
- pagination
- selection of any font available to the output device
- adjustment of type size and vertical spacing (or "leading")
- configuration of line length and indentation amounts; columnation
- drawing of geometric primitives (lines, arcs, polygons, circles, . . .)
- setup of stroke and fill colors (where supported by the output device)
- embedding of hyperlinks, images, document metadata, and other inclusions (where supported by the output device)

`roff` as a programming language also supports evaluation of arithmetic expressions, storage and retrieval of parameters ("registers" and "strings"), and altered flow of control dependent on the foregoing.

1.4 Macro Packages

Elemental typesetting functions can be be laborious to use directly with complex documents. A *macro* facility specifies how certain routine opera-

tions, such as starting paragraphs, or printing headers and footers, should be performed in terms of those low-level instructions. One then *calls* the macro to make it perform its task. Macros can be specific to one document or collected together into a *macro package* for use by many. `groff` supplies versions of the widely used macro packages `man`, `mdoc`, `me`, `mm`, `mom`, and `ms`.

1.5 Preprocessors

An alternative approach to complexity management, particularly when constructing tables, setting mathematics, or drawing diagrams, lies in preprocessing. A *preprocessor* employs a domian-specific language to ease the generation of tables, equations, and so forth in terms that are convenient for human entry. Each preprocessor reads a document and translates relevant portions of it into GNU `troff` input. Command-line options to `groff` tell it which preprocessors to use.

`groff` provides preprocessors for laying out tables (`tbl`), typesetting equations (`eqn`), drawing diagrams (`pic` and `grn`), inserting bibliographic references (`refer`), and drawing chemical structures (`chem`). An associated program that is useful when dealing with preprocessors is `soelim`.

`groff` also supports `grap`, a preprocessor for drawing graphs. A free implementation of it can be obtained separately.[1]

Unique to `groff` is the `preconv` preprocessor that enables GNU `troff` to handle documents in a variety of input encodings, including UTF-8. Unlike most preprocessors, `preconv` operates on its entire enput rather than transforming specially marked regions of a document.

Other preprocessors exist, but no free implementations are known. An example is `ideal`, which draws diagrams using a mathematical constraint language.

1.6 Output Devices

GNU `troff`'s output is in a device-independent page description language. An *output driver* translates this language into a file format or byte stream that a piece of (possibly emulated) hardware understands. `groff` features output drivers for PostScript devices, terminal emulators (and other simple typewriter-like machines), X11 (for previewing), TEX DVI, HP LaserJet 4/PCL5 printers, Canon LBP (CaPSL-using printers), HTML, XHTML, and PDF.

1.7 Installation

Locate installation instructions in the files `INSTALL`, `INSTALL.extra`, and `INSTALL.REPO` in the `groff` source distribution. Being a GNU project, `groff` supports the familiar '`./configure && make`' command sequence.

[1] `https://www.lunabase.org/~faber/Vault/software/grap/`

1.8 Conventions Used in This Manual

We apply the term "groff" to the language documented here, the GNU implementation of the overall system, the project that develops that system, and the command of that name. In the first sense, `groff` is an extended dialect of the `roff` language, for which many similar implementations exist. We say "the formatter" when speaking of behavior that is generally true of `troff` and `nroff` programs.

A tradition has arisen that GNU programs' names bear a prefix '`g`' where necessary to distinguish them from other implementations on the host system (see Section 2.2 [Environment], page 13). Thus, for example, '`geqn`' is GNU `eqn`. On operating systems that lack a `troff` of different provenance, this prefix is omitted; GNU `troff` is the only `troff` available. Exceptionally, '`groff`' always retains its leading '`g`'.

We call non-GNU `troff` systems AT&T `troff` because that is the common origin of almost all other `troff` implementations[2] (with more or less compatible changes).

This manual employs Emacs names for non-graphic keycap engravings on the alphabetic section of the keyboard. "`RET`" is Return or Enter, and "`SPC`" is the space bar.

The `roff` language features several major syntactical categories within which many items are predefined. Presentations of these items comprise the form in which the item is most commonly used on the left, and, aligned to the right margin, the name of the category in brackets.

`\n[example]` [Register]

The register '`example`' is one that that `groff` *doesn't* predefine. You can create it yourself, though; see Section 5.8.1 [Setting Registers], page 99.

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

We use Texinfo's "result" (⇒) and `error` notations to present output written to the standard output and standard error streams, respectively. Diagnostic messages from the GNU `troff` formatter and other programs are examples of the latter, but the formatter can also be directed to write user-specified messages to the standard error stream. The notation then serves to identify the output stream and does not necessarily mean that an error has occurred.[3]

---

[2] Besides `groff`, `neatroff` is an exception.

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

```
$ echo "Twelve o'clock and" | groff -T ascii | sed '/^$/d'
    ⇒ Twelve o'clock and
$ echo '.tm all is well.' | groff > /dev/null
    error  all is well.
```

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

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

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

We encourage you to to try the examples yourself, and to alter them to better learn `groff`'s behavior. Our examples sometimes need to direct the formatter to set a line length (with '`.ll`') that fits within the page margins of this manual. We mention this so that you know why it is there before we discuss the `ll` request formally.[4]

We refer occasionally to *man pages*, in which aspects of the `groff` system or of its operating environment are further documented.[5] When you see a citation like *groff_man*(7), understand that you can type '`man groff_man`' at the command line to view it. The numbered category distinguishes pages by their purpose. You can try '`man 'groff(1)'`' and '`man 'groff(7)'`' to observe this distinction.[6] Your system likely offers an *intro*(1) page that will help you make the most of this resource.

1.9 Credits

We adapted portions of this manual from existing documents. James Clark's man pages were an invaluable foundation; we have updated them in parallel with the development of this manual. We based the tutorial for macro package users on Eric Allman's introduction to his `me` macro package (which we also provide, little altered from 4.4BSD). Larry Kollar contributed much of the material on the `ms` macro package.

---

[4] See Section 5.15 [Line Layout], page 131.

[5] `roff` is the language of historical Unix manuals, and of man pages to this day.

[6] POSIX has not standardized a mechanism for the `man` command to distinguish pages by numeric category. If '`man 'groff(7)'`' produces an error, attempt '`man 7 groff`' or '`man -s 7 groff`'.

2 Invoking groff

This chapter focuses on how to invoke the groff front end, which constructs a pipeline connecting desired preprocessors, the GNU troff formatter program, and a postprocessor.

2.1 Options

groff runs the GNU troff program and, normally, a postprocessor appropriate to the selected device. The default device is 'ps', unless changed at groff's build-time configuration. groff can preprocess input with any of pic, eqn, tbl, grn, grap, chem, refer, soelim, or preconv.

This section documents only options to the groff front end. Since it passes many of its arguments to GNU troff, we describe many of the latter's options here. Arguments to preprocessors and output drivers can be found in the man pages *pic*(1), *eqn*(1), *tbl*(1), *grn*(1), *refer*(1), *chem*(1), *soelim*(1), *preconv*(1), *grotty*(1), *grops*(1), *gropdf*(1), *grohtml*(1), *grodvi*(1), *grolj4*(1), *grolbp*(1), and *gxditview*(1).

A summary of groff's usage follows.

```
groff [-abcCeEgGijklNpRsStUVXzZ] [-d cs] [-d string=text]
      [-D fallback-encoding] [-f font-family]
      [-F font-directory] [-I inclusion-directory]
      [-K input-encoding] [-L spooler-argument]
      [-m macro-package] [-M macro-directory]
      [-n page-number] [-o page-list]
      [-P postprocessor-argument] [-r cnumeric-expression]
      [-r register=numeric-expression] [-T output-device]
      [-w warning-category] [-W warning-category]
      [file ...]
```

troff shares much of this interface; groff passes relevant options and operands to it.

```
troff [-abcCEiRSUz] [-f font-family] [-F font-directory]
      [-I inclusion-directory] [-m macro-package]
      [-M macro-directory] [-n page-number] [-o page-list]
      [-r cnumeric-expression]
      [-r register=numeric-expression] [-T output-device]
      [-w warning-category] [-W warning-category]
      [file ...]
```

Options that don't take arguments can be clustered after a single -. A *file* operand of - denotes the standard input stream.

All groff commands accept a --help option, which summarizes usage similarly to the foregoing, and --version, which discloses release information. Both exit with a successful status after reporting.

The rest of groff's command-line options are as follows.

'-a' Generate a plain text approximation of the typeset output. The read-only register .A is set to 1. See Section 5.8.5 [Built-in Registers], page 105. 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 formatted output.
- Vertical motion, apart from that implied by a break, is not represented.
- A horizontal motion of any size is represented as one space. Adjacent horizontal motions are not combined. Supplemental inter-sentence space (configured by the second argument to the ss request) is not represented.
- A special character is rendered as its identifier between angle brackets; for example, a hyphen appears as '<hy>'.

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

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

'-c' Disable multi-color output and color request's ability to enable it.

'-C' Enable AT&T troff compatibility mode; implies -c. See Section 5.39 [Implementation Differences], page 248, for the list of incompatibilities between groff and AT&T troff.

'-d *ctext*'
'-d *string*=*text*'
Define roff string *c* or *string* as *text*. *c* must be one character; *string* can be of arbitrary length. Such assignments happen before any macro file is loaded, including the startup file. Due to *getopt_long*(3) limitations, *c* cannot be, and *string* cannot contain, an equals sign, even though that is a valid character in a roff identifier. See Section 5.22 [Strings], page 170.

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

'-e' Run eqn preprocessor.

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

'-f *fam*' Use *fam* as the default font family. See Section 5.19.2 [Font Families], page 143.

'-F *dir*' Search in directory *dir* for the selected output device's directory of device and font description files. See Section 2.4 [Font Directories], page 15.

'-g' Run grn preprocessor.

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

'-h' Display a usage message and exit.

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

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

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

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

groff passes -I options and their arguments to grn, soelim, troff, and output drivers.

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

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

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

'-l' Send the output to a spooler for printing. The print directive in the device description file specifies the default command to be used; see Section 6.1 [Device and Font Description Files], page 257. See options -L and -X.

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

'-m *mac*' Search for the macro package *mac*.tmac and read it prior to any input. If not found, tmac.*mac* is attempted. See Section 2.3 [Macro Directories], page 14. groff passes -m options and their arguments to eqn, grap, and grn.

The following "full-service" macro packages are available. *groff_tmac*(5) documents these and many other, "auxiliary", packages.[1]

Two full-service macro packages are specialized for formatting Unix reference manuals; they do not support features like footnotes or multiple columnation.

man constructs man pages in a format introduced by Seventh Edition Unix (1979). Its macro interface is small, and the package widely used; see *groff_man*(7).

mdoc constructs man pages in a format introduced by 4.3BSD-Reno (1990). It provides many more features than man but is also larger, more complex, and not as widely adopted; see *groff_mdoc*(7).

mandoc recognizes a document's use of man or mdoc and loads the corresponding macro package. Multiple man pages, in either format, can be handled; mandoc reloads each macro package as necessary.

The following packages each support composition of documents of any kind, from single-page memos to lengthy monographs. They are similar in functionality; select one that suits your taste.

me originates in 2BSD (1978); see *groff_me*(7).

mm originates in Programmer's Workbench (PWB) Unix 1.0 (1977); see *groff_mm*(7).

mom was contributed to groff in 2002, and freely exercises its many extended features. see *groff_mom*(7).

ms originates in Sixth Edition Unix (1975); see *groff_ms*(7).

'-M *dir*' Search directory *dir* for macro files. See Section 2.3 [Macro Directories], page 14. groff passes -M options and their arguments to eqn, grap, and grn.

'-n *num*' Begin numbering pages at *num*. The default is '1'.

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

[1] See Section 5.1.8 [Macro Packages], page 74.

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

'-p' Run pic preprocessor.

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

'-r *cnumeric-expression*'
'-r *register=numeric-expression*'
Define roff register *c* or *register* as *numeric-expression* (see Section 5.4 [Numeric Expressions], page 82). *c* must be one character; *register* can be of arbitrary length. Such assignments happen before any macro file is loaded, including the startup file. Due to *getopt_long*(3) limitations, *c* cannot be, and *register* cannot contain, an equals sign, even though that is a valid character in a roff identifier. See Section 5.8 [Registers], page 99.

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

troff also accepts a -R option, which is not accessible via groff. This option facilitates troubleshooting by preventing the loading of the troffrc and troffrc-end files.

'-s' Run soelim preprocessor.

'-S' Operate in "safer" mode; see -U below for its opposite. Safer mode is enabled by default. Explicitly specifying -S causes troff to ignore any subsequent -U option.

'-t' Run tbl preprocessor.

'-T *dev*' Prepare output for device *dev*. groff passes the -T option and its argument to troff, then (unless the -Z option is used) runs an output driver to convert troff's output to a form appropriate for *dev*. The following output devices are available.

ps For PostScript printers and previewers.

pdf For PDF viewers or printers.

dvi For TEX DVI format.

X75 For a 75 dpi X11 previewer.

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

`X100` For a 100 dpi X11 previewer.

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

`ascii` For typewriter-like devices using the (7-bit) ISO 646:1991 IRV (US-ASCII) character set.

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

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

`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 formatter string `.T` contains the name of the selected 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 run GNU `troff`). See Section 5.8.5 [Built-in Registers], page 105.

The device description file's `postpro` command specifies the default postprocessor to be used by a device. See Section 6.1 [Device and Font Description Files], page 257. The `-X` option overriddes this selection.

'`-U`' Operate in *unsafe mode*, enabling the `cf`, `open`, `opena`, `pi`, `pso`, and `sy` requests, which are disabled by default because they allow an untrusted input document to run arbitrary commands, put arbitrary content into `troff` output, or write to arbitrary file names.[2] This option also adds the current directory to the macro package search path; see the `-m` and `-M` option above. `groff` passes `-U` to `pic` and GNU `troff`.

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

[2] GNU `troff` does not, however, accept newlines (line feeds) in file names supplied as arguments to requests.

'-V' Write the command pipeline that groff would run to the standard output stream and exit. If given more than once, groff both writes the pipeline to the standard error stream and runs it.

'-w *cat*'
'-W *cat*' Enable and inhibit, respectively, warnings in category *cat*. See Section 5.38.1 [Warnings], page 245.

'-X' Use gxditview instead of the usual postprocessor to (pre)view a document on an X11 display. Combining this option with -T ps uses the font metrics of the PostScript device, whereas the -T X75, -T X75-12, -T X100, and -T X100-12 options use the metrics of X11 fonts.

'-z' Suppress formatted output from troff.

'-Z' Disable postprocessing. troff output appears on the standard output stream (unless suppressed with -z); see Section 6.2 [GNU troff Output], page 263, for a description of this format.

2.2 Environment

Environment variables in the host system affect the behavior of programs supplied by groff as follows. Normally, the path separator in environment variables ending with 'PATH' is the colon; this may vary depending on the operating system. For example, Windows uses a semicolon instead.

GROFF_BIN_PATH
: Locate groff commands in these directories, followed by those in PATH. If not set, the installation directory of GNU roff executables, documented in *groff*(1), is searched before PATH.

GROFF_COMMAND_PREFIX
: Apply a prefix to certain GNU roff commands. groff can be configured at compile time to apply a prefix to the names of programs it provides that had counterparts in AT&T troff, so that name collisions are avoided at run time. The default prefix is empty.

When used, this prefix is conventionally the letter 'g'. For example, GNU troff would be installed as troff. Besides troff, the prefix applies to the formatter wrapper nroff; the preprocessors eqn, grn, pic, refer, tbl, and soelim; and the utilities indxbib and lookbib.

GROFF_ENCODING
: Specify the assumed character encoding of the input. groff passes its value as an argument to the preconv preprocessor's -e option. This variable's existence implies the groff option

`-k`. If set but empty, `groff` runs `preconv` without an `-e` option. `groff`'s `-K` option overrides `GROFF_ENCODING`. See *preconv*(7).

`GROFF_FONT_PATH`
: Seek the selected output device's directory of device and font description files in this list of directories. See Section 2.4 [Font Directories], page 15, *troff*(1), and *groff_font*(5).

`GROFF_TMAC_PATH`
: Seek macro packages in this list of directories. See Section 2.3 [Macro Directories], page 14, *troff*(1), and *groff_tmac*(5).

`GROFF_TMPDIR`
: Create temporary files in this directory. If not set, but `TMPDIR` is, the latter is used instead. On Windows systems, if neither of the foregoing is set, the environment variables `TMP` and `TEMP` (in that order) are checked also. Otherwise, temporary files are created in a system-dependent default directory (on Unix and GNU/Linux systems, usually `/tmp`). The `refer`, `grohtml`, and `grops` commands use temporary files.

`GROFF_TYPESETTER`
: Set the default output device. The `-T` *`dev`* option overrides it. If empty or unset, a default configured at build time, and documented in *groff*(1), is used.

`SOURCE_DATE_EPOCH`
: Declare a time stamp (expressed as seconds since the Unix epoch) to use as the output creation time stamp in place of the current time. The time is converted to human-readable form using *gmtime*(3) and *asctime*(3) when the formatter starts up and stored in registers usable by documents and macro packages (see Section 5.8.5 [Built-in Registers], page 105).

`TZ`
: Declare the time zone to use when converting the current time to human-readable form; see *tzset*(3). If `SOURCE_DATE_EPOCH` is used, it is always converted to human-readable form using UTC.

2.3 Macro Directories

A macro file must have a name in the form *`name`*`.tmac` or `tmac.`*`name`* and be placed in a *tmac directory* to be found by the `-m` *`mac`* command-line option.[3] Such naming and placement makes a macro file into a *macro package*; when requested, it is sought in several directories. Together, these locations constitute the *tmac path*. Each directory is searched in the following order until the desired package is found or the list is exhausted.

[3] The `mso` request loads a macro file of any name. See Section 5.34 [Host System Service Access], page 228.

- Directories specified with the `-M` command-line option.
- Directories listed in the `GROFF_TMAC_PATH` environment variable.
- The current working directory (only if in unsafe mode using the `-U` command-line option).
- The user's home directory, found in the `HOME` environment variable.
- A site-local platform-dependent directory, a site-local platform-independent directory, and a stock directory. Locations corresponding to your installation are listed in section "Environment" of *troff*(1). If not otherwise configured, they are as follows.

  ```
  /usr/local/lib/groff/site-tmac
  /usr/local/share/groff/site-tmac
  /usr/local/share/groff/1.23.0/tmac
  ```

  The foregoing assumes that the version of `groff` is 1.23.0, and that the installation prefix was `/usr/local`. These locations can be customized as part of the build-time configuration process.

2.4 Font Directories

The GNU `troff` formatter and `groff`'s output drivers read *device* and *font description files* that detail the output device and the typefaces available to it, including their glyph repertoires and the *metrics* (dimensions) of each glyph. This information permits the formatter to accurately place glyphs with respect to each other. The device description file is always named `DESC`; fonts are typically described in files with short names like `TR`, `CR`, `HBI`, or `S`.[4]

Device and font description files are kept in *font directories*, which together constitute the *font path*. The search procedure always appends the directory `dev`*name*, where *name* is the name of the output device. Assuming TEX DVI output, and `/foo/bar` as a font directory, the description files for `grodvi` must be in `/foo/bar/devdvi`. Each directory in the font path is searched in the following order until the desired description file is found or the list is exhausted.

- Directories specified with the `-f` command-line option. All output drivers (and some preprocessors) support this option as well, because they require information about glyphs to be rendered in the document.
- Directories listed in the `GROFF_FONT_PATH` environment variable.
- A site-local directory and a stock directory. Locations corresponding to your installation are listed in section "Environment" of *troff*(1). If not otherwise configured, they are as follows.

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

[4] See Section 6.1 [Device and Font Description Files], page 257.

The foregoing assumes that the version of `groff` is 1.23.0, and that the installation prefix was `/usr/local`. These locations can be customized as part of the build-time configuration process.

2.5 Paper Format

The formatter reads the device description file `DESC` for the selected output device when it starts; page dimensions declared there are used if present.

`groff`'s build process configures a default page format and writes it to typesetters' `DESC` files. This installation defaults to '`letter`'. If the `DESC` file lacks this information, the formatter and output driver use a page length of '`11i`' (eleven inches) for compatibility with AT&T `troff`.

In the formatter, the `pl` request changes the page length, but macro packages often do not support alteration of the paper format within a document. One might, for instance, want to switch between portrait and landscape orientations. Macro packages lack a consistent approach to configuration of parameters dependent on the paper format; some, like `ms`, benefit from a preamble in the document prior to the first macro call, while others, like `mm`, instead require the specification of registers on the command line, or otherwise before its macro file is interpreted, to configure page dimensions.

Output drivers for typesetters also recognize command-line options `-p` to override the default page dimensions and `-l` to use landscape orientation. The output driver's man page, such as *grops*(1), may be helpful.

`groff`'s `-d paper` command-line option is a convenient means of setting the paper format; see *groff_tmac*(5). Combine it with appropriate `-P` options for the output driver, overriding its defaults. The following command formats for PostScript on A4 paper in landscape orientation.

```
$ groff -T ps -d paper=a4l -P -pa4 -P -l -m s my.ms >my.ps
```

2.6 Invocation Examples

`roff` systems are best known for formatting man pages. A `man` librarian program, having located a page, might render it with a `groff` command.

```
$ groff -t -m an -T utf8 /usr/share/man/man1/groff.1
```

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

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

```
$ pic foo.me | tbl | troff -m e -T utf8 | grotty
```

Shorten this pipeline to an equivalent command using `groff`.

```
$ groff -p -t -m e -T utf8 foo.me
```

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

```
$ $(grog -T utf8 foo.me)
```

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

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

and

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

are equivalent.

3 Tutorial for Macro Package Users

Most users of the `roff` language employ a macro package to format their documents. Successful macro packages ease the composition process; their users need not master the full formatting language, nor understand features like diversions, traps, and environments. This chapter aims to familiarize you with basic concepts and mechanisms common to many macro packages (like "displays"). If you prefer a meticulous and comprehensive presentation of the language and its formatter, peruse Chapter 5 [GNU `troff` Reference], page 67, instead.

3.1 Basics

Let us first survey some basic concepts necessary to use a macro package fruitfully.[1] References are made throughout to more detailed information.

GNU `troff` reads input prepared by the user and outputs a formatted document suitable for publication or framing. The input consists of text, or words to be printed, and embedded commands (*requests* and *escape sequences*), which tell GNU `troff` how to format the output. See Section 5.6 [Formatter Instructions], page 89.

The primary function of GNU `troff` is to collect words from its input, *fill* output lines with those words, *break* the line at or near the right-hand margin (possibly by hyphenating a word), *adjust* the line to reach that margin (if necessary) by widening spaces between words, and output the result.

```
In fact, we know full well today that it is futile to
speak of liberty as long as economic slavery exists.
(Kropotkin)
  ⇒ In  fact,  we know full well today that it
  ⇒ is futile to speak of liberty as  long  as
  ⇒ economic slavery exists.  (Kropotkin)
```

Sometimes a new output line should start even though the current line is not yet full—for example, at the end of a paragraph. GNU `troff` will do this for us automatically at the end of input, but we often want a break sooner, and more frequently. We wish to *instruct* the formatter.

To that end, not all input lines are *text lines* containing words to be formatted. *Control lines* start with a dot ('`.`') or an apostrophe ('`'`') as the first character, and are followed by a request or macro name that tells a macro package (or GNU `troff` directly) how to format the text.

We can command a break with the `br` request. Some requests cause a break automatically, as do (normally) blank input lines and input lines beginning with a space or tab.

---

[1] The remainder of this chapter is based on "Writing Papers with NROFF using -me" by Eric P. Allman, which is distributed with `groff` as `meintro.me`.

A *macro* bundles text and/or control lines into a named collection that can be called like a request. A macro can also be called by a *trap* that is set to "go off" automatically at certain places on the page. Thus, while requests perform primitive operations, macros handle complex ones, like arranging the output into columns, collecting and writing out footnotes, or managing page headers and footers.

Many requests and macros accept *arguments* that influence their behavior. A "plain" `sp` request breaks and puts a blank line on the output. But

```
.sp 4
```

spaces four lines instead. Spaces (but *not* tabs) separate arguments from the request or macro name and from each other.

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

- First, keep the input lines short. Short input lines are easier to edit, and, when filling, GNU `troff` packs words onto longer lines anyhow.
- Second, it is helpful to begin a new line after every sentence, comma, semicolon, or colon, since common revisions are to add, delete, or replace sentences, clauses, phrases, or members of lists.
- If you *don't* start a sentence on a new line, put two spaces after the previous sentence. GNU `troff` then recognizes punctuation that ends a sentence, and inserts inter-sentence space accordingly.

We offer further advice in Section 5.1.11 [Input Conventions], page 76.

Vertical spacing is the distance between lines of text; it is expressed in the same units as the type size—the point. The default is 10-point type on 12-point spacing. To get *double-spaced* text you would set the vertical spacing to 24 points. Some, but not all, macro packages expose a macro or register to configure the vertical spacing.

A number of requests allow you to change the way the output is arranged on the page, sometimes called its *layout*. Most macro packages don't supply macros for performing these (at least not without performing other actions besides), as they are such basic operations. The macro packages for writing man pages, `man` and `mdoc`, discourage explicit use of these requests altogether.

Arguments to requests and macro calls can often be *measurements* rather than simple integers. For instance,

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

outputs one and a half inches of vertical space, followed by the line "My thoughts on the subject", followed by a single blank line (more measurement units are available; see Section 5.3 [Measurements], page 80). Excess vertical space is normally discarded at page or column breaks. If the above example appears one inch from the bottom of the page, the half inch of space "left over" does not appear at the top of the next.

If you desire precise spacing control when using a macro package, be advised that it might not honor `sp` requests as you expect; it can use a formatter feature called *no-space mode* to prevent excess space from accumulating. See Section 5.11 [Manipulating Spacing], page 122. Use the facilities the package offers to control spacing between paragraphs, before section headings, and around displays (discussed below).

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

```
.ce 1000
up to one thousand lines of input
.ce 0
```

The '`.ce 0`' request tells GNU `troff` to center zero more text lines—in other words, to stop centering.

GNU `troff` also offers the `rj` request for right-aligning text. It works analogously to `ce` and is convenient for setting epigraphs.

The `bp` request starts a new page.

All of these requests cause a break, starting a new line. If you invoke them with the apostrophe '`'`', the *no-break control character*, the (initial) break they normally perform is suppressed. '`'br`' does nothing.

3.2 Common Features

GNU `troff` provides low-level operations for formatting a document. Many routine operations are undertaken in nearly all documents that require a series of such primitive operations to be performed. These common tasks are grouped into *macros*, which are then collected into a *macro package*.

Some macro packages ("major" or "full-service") assume responsibility for page layout and other critical functions; others ("supplemental" or "auxiliary") do not.

We present several capabilities of full-service macro packages below. Each package employs its own macro names to exercise them. For details, consult the package's man page or, for `ms`, see Section 4.6 [ms], page 28.

3.2.1 Paragraphs

Paragraphs can be formatted in various ways. Some indent their first line. Block paragraphs like the following example omit this indentation, and must be separated with vertical space for readability. Separation can be configured for other paragraph types as well.

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

We also frequently encounter *tagged* paragraphs, which begin with a label, or *tag*, at the left margin, and indent the remaining text.

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

If the tag is too wide for the indentation amount, the line is broken.

```
⇒ longlabel
⇒      The long tag does not align with subsequent
⇒      lines, but those lines align with each other.
```

A variation of the tagged paragraph is the itemized or enumerated paragraph, which might use punctuation or a digit for a tag, respectively. These are frequently used to construct lists.

```
⇒ *    This list item starts with a bullet.  If a bullet
⇒      glyph  is unavailable, groff produces an asterisk
⇒      instead.
```

Often, use of the same macro without a tag continues such a discussion.

```
⇒ -xyz  This option is recognized but ignored.
⇒
⇒       It had a security hole that we don't discuss.
```

3.2.2 Sections and Chapters

A simple kind of section heading is unnumbered, set in a bold or italic style, and occupies a line by itself. Others possess automatically numbered multi-level headings and/or different typeface styles or sizes at different levels. More sophisticated macro packages supply macros for designating chapters and appendices, and permit *run-in headings*, where there is no break between the end of the heading text and the start of the subsequent paragraph.

3.2.3 Headers and Footers

Headers and *footers* occupy the top and bottom of each page, respectively, and contain data like the page number and the article or chapter title. Their appearance is not affected by the running text.

Headers and footers are together called *titles*, and comprise three parts: left-aligned, centered, and right-aligned. The formatter replaces a '`%`' character appearing anywhere in a title with the page number. See Section 5.17 [Page Layout], page 136. Some packages allow for different titles on even- and odd-numbered pages (for printed, bound material).

3.2.4 Page Layout

Most macro packages let the user specify the size of the page margins. The top and bottom margins are typically handled differently than the left and right margins; the latter are derived from the *page offset*, *indentation*, and *line length*. See Section 5.15 [Line Layout], page 131. Commonly, packages support registers to tune these values.

3.2.5 Displays and Keeps

Displays are sections of text set off from the surrounding material (typically paragraphs), often differing in indentation and/or spacing. Tables, block quotations, and figures are displayed. Equations and code examples, when not much shorter than an output line, often are. Lists may or may not be.

A *keep* is a group of output lines, often a display, that is formatted on a single page if possible; it causes a page break to happen early if necessary to not interrupt the kept material. Packages for setting man pages support example displays but not keeps.

Floating keeps can move, or "float", relative to the text around them in the input. They are useful for displays that are captioned and referred to by name, as with "See figure 3". A floating keep might appear at the bottom of the current page if it fits, and at the top of the next otherwise. Alternatively, it might be deferred to the end of a section. Use of a floating keep can prevent a large vertical space from appearing before a tall keep of the ordinary sort when it won't fit on the page.

3.2.6 Footnotes and Endnotes

Footnotes and *endnotes* are forms of delayed formatting. They are recorded at their points of relevance in the input, but not formatted there. Instead, a *mark* cues the reader to check the "foot", or bottom, of the current page, or in the case of endnotes, an annotation list later in the document. Macro packages that support these features also supply a means of automatically numbering either type of annotation.

3.2.7 Table of Contents

A package may handle a *table of contents* by directing section heading macros to save the heading's text and the page number where it occurs for use in a later *entry* for a table of contents. It writes the collected entries at the end of the document, once all are known, upon request. A *leader*, a row of dots, bridges the text on the left with its location on the right. Other collections might work in this manner, providing lists of figures or tables.

A table of contents is often found at the end of a GNU `troff` document because the formatter processes the document in a single pass. The `gropdf` output driver supports a PDF feature that relocates pages at the time the document is rendered; see *gropdf*(1).

3.2.8 Indexing

An index is similar to a table of contents, in that entry labels and locations must be collected, but poses a greater challenge because it needs to be sorted before it is output. Here, processing the document in multiple passes is inescapable, and tools like the *makeindex*(1) program become necessary.

3.2.9 Document Formats

Some macro packages supply stock configurations of certain types of documents, like business letters and memoranda. These often also have provision for a *cover sheet*, which may be rigid in its format. With these features, it is even more important to use the package's macros in preference to the formatter requests presented earlier, where possible.

3.2.10 Columnation

Macro packages apart from `man` and `mdoc` for man page formatting offer a facility for setting multiple text columns on the page.

3.2.11 Font and Size Changes

The formatter's requests and escape sequences for setting the typeface and size are not always intuitive in their behavior, so all full-service packages provide macros to simplify input of these operations. They can also make mid-word font style changes more convenient, and can handle italic corrections automatically. See Section 5.19.9 [Italic Corrections], page 160.

3.2.12 Predefined Text

Most macro packages supply predefined strings to set computed text like the date, or to perform operations like super- and subscripting.

3.2.13 Preprocessor Support

All macro packages provide support for various preprocessors and may extend their functionality by defining macros to caption their output and/or set it in a display. Examples include `TS` and `TE` for `tbl`, `EQ` and `EN` for `eqn`, and `PS` and `PE` for `pic`. Another preprocessor, `refer`, facilitates the inclusion of bibliographic citations in a consistent format.

3.2.14 Configuration and Customization

Each package provides means of customizing details of its behavior. Often, this is achieved with register and string definitions. Such parameters include the default type size and the appearance of section headings.

4 Macro Packages

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

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

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

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

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

Many auxiliary, or supplemental, macro packages are also available. They may in general be used with any full-service macro package and handle a variety of tasks from character encoding selection, to language localization, to inlining of raster images. See *groff_tmac*(5) for a list.

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 *groff_man*(7).

4.1.1 Optional man extensions

Use the file `man.local` to configure its rendering parameters on a persistent basis. With care, its macros can be redefined there (except for `TH`, to which one should, at most, append with the `am` family of requests).

Custom headers and footers

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

`.PT` [Macro]

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

`.BT` [Macro]

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

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

Ultrix-specific man macros

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

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

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

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

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

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

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

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

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

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

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

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

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

note is printed centered (the second argument replaces the word "Note" if specified).

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

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

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

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

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

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

`.VE` [Macro]
End printing the change bar begun by `VS`.

Simple example

The following example `man.local` file alters the behavior of the `SH` macro.

```
.\" Make the heading font Helvetica bold.
.ds HF HB
.
.\" Add vertical space prior to headings on typesetters.
.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 *groff_mdoc*(7).

Use the file `mdoc.local` to configure its rendering parameters on a persistent basis. With care, its macros can be redefined there (except for `Dd`, to which one should, at most, append with the `am` family of requests).

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. *groff_me*(7) identifies the installation path for these documents.

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

4.4 mm

groff's implementation of the AT&T memorandum macro package is documented in *groff_mm*(7).

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

4.5 mom

The mom package's primary documentation is in HTML. Model documents illustrating many features are offered in PDF. See the *groff*(1) man page, section "Installation Directories", for their location.

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

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

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

4.6 ms

Use the ms ("manuscript") package to compose letters, memoranda, reports, and books. These groff macros feature cover page and table of contents generation, automatically numbered headings, several paragraph styles, a variety of text styling options, footnotes, multi-column page layouts, and support for PDF features when using '-T pdf'. ms supports the tbl, eqn, pic, and refer preprocessors for inclusion of tables, mathematical equations, diagrams, and consistently formatted bibliographic citations. groff ms is mostly compatible with the documented interface and behavior of AT&T Unix Version 7 ms. It recreates most extensions from 4.2BSD (Berkeley) and Research Tenth Edition Unix.

4.6.1 Introduction

The `ms` macros are the oldest surviving package for `roff` systems.[1] Whereas `man` suits brief references, `ms` can handle long or complex works that require deep sectioning, footnotes, or multiple columnation.

Macro, register, and string descriptions frequently mention each other; most references are to macros. Where a register or string is referenced, we annotate its type. `ms`'s identifiers use only capital letters, numerals, and '`-`'. Macro arguments surrounded by [brackets] are optional. Setting the register `PDFFEAT` to 1 on the command line enables PDF features.

4.6.1.1 Basic information

Prepare an `ms` document with your preferred text editor. Call an `ms` macro early in the document to initialize the package. A *macro* is a formatting instruction to `ms`; to call it, start a line with a dot followed by its name. Use '`.PP`' if you want your paragraph's first line indented, or '`.LP`' if you don't. Then type text normally. It is a good practice to start each sentence on a new line, or to put two spaces after sentence-ending punctuation, so that the formatter knows where the sentence boundaries are. You can separate paragraphs with further paragraphing macros, or with blank lines, and you can indent with tabs. When you need one of the features mentioned earlier (see Section 4.6 [ms], page 28), return to this subsection.

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

```
$ editor radical.ms # vim, emacs, nano, ...
$ nroff -ww -z -ms radical.ms # check for errors
$ nroff -ms radical.ms | less -R
$ groff -T ps -ms radical.ms > radical.ps
$ see radical.ps # or your favorite PDF viewer
```

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

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

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

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

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

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

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

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

In the foregoing, we see that `\"` begins a comment. This is an example of an *escape sequence*, the other kind of formatting instruction. Escape sequences can appear almost anywhere. They begin with the escape character (`\`) and are followed by at least one more character. `ms` documents tend to use only a few of `groff`'s many requests and escape sequences; see Section B.1 [Request Index], page 301, and Section B.2 [Escape Sequence Index], page 304, or the *groff*(7) man page for complete lists.

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

Prefix any words that start with a dot '`.`' or neutral apostrophe '`'`' with `\&` if they are at the beginning of an input line (or might become that

way in editing) to prevent them from being interpreted as macro calls or requests. Suffix '.', '?', and '!' with \& when needed to cancel end-of-sentence detection.

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

4.6.2 Document Structure

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

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

Format and layout
: By setting registers and strings, you can configure your document's typeface, margins, spacing, headers and footers, and footnote arrangement. See Section 4.6.3 [ms Document Control Settings], page 32.

Document description
: A document description consists of any of: a title, one or more authors' names and affiliated institutions, an abstract, and a date or other identifier. See Section 4.6.4 [ms Document Description Macros], page 38.

Body text
: The main matter of your document follows its description (if any). ms supports highly structured text consisting of paragraphs interspersed with multi-level headings (chapters, sections, subsections, and so forth) and augmented by lists, footnotes, tables, diagrams, and similar material. See Section 4.6.5 [ms Body Text], page 39.

Tables of contents
: Macros enable the collection of entries for a table of contents (or index) as the material they discuss appears in the document. A macro call at the end of the document emits the collected entries. This material necessarily follows the rest of the text since troff is a single-pass formatter; it cannot determine the page number of a division of the text until it has been set and output. Since ms output was designed for the production of hard copy, the traditional procedure was to manually relocate the pages containing the table of contents between the cover page

and the body text. Today, page resequencing is more often done in the digital domain; if you mandate a separate cover page by calling `RP` and use the `pdf` output device, `groff ms` positions the table of contents immediately after this cover page. An index works similarly, but because it typically needs to be sorted after collection, its preparation requires separate processing.

4.6.3 Document Control Settings

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

`.nr` *reg value* [Request]
Set register *reg* to *value*.

`.ds` *name contents* [Request]
Set string *name* to *contents*.

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

Margin settings

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

Effective: next page.

Default: Varies by output device and paper format; 1 i is used for typesetters using U.S. letter paper, and zero for terminals. See Section 2.5 [Paper Format], page 16.

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

Effective: next paragraph.

Default: Varies by output device and paper format; 6.5 i is used for typesetters using U.S. letter paper (see Section 2.5 [Paper Format], page 16) and 65 n on terminals.

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

Effective: next paragraph.

Default: Varies by output device and paper format; 6.5 i is used for typesetters using U.S. letter paper (see Section 2.5 [Paper Format], page 16) and 65 n on terminals.

`\n[HM]` [Register]
Defines the header margin height at the top of the page.
Effective: next page.
Default: 1 i.

`\n[FM]` [Register]
Defines the footer margin height at the bottom of the page.
Effective: next page.
Default: 1 i.

Titles (headers, footers)

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

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

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

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

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

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

Text settings

`\n[PS]` [Register]

Defines the type size of the body text.

Effective: next paragraph.

Default: 10 p.

`\n[VS]` [Register]

Defines the vertical spacing (type size plus leading).

Effective: next paragraph.

Default: 12 p.

`\n[HY]` [Register]

Defines the automatic hyphenation mode used with the `hy` request. Setting `HY` to 0 disables automatic hyphenation. This is a Research Tenth Edition Unix extension.

Effective: next paragraph.

Default: 6.

`\*[FAM]` [String]

Defines the font family used to typeset the document. This is a GNU extension.

Effective: next paragraph.

Default: defined by the output device; often '`T`' (see Section 4.6.5 [ms Body Text], page 39)

Paragraph settings

`\n[PI]` [Register]

Defines the indentation amount used by the `PP`, `IP` (unless overridden by an optional argument), `XP`, and `RS` macros.

Effective: next paragraph.

Default: 5 n.

`\n[PD]` [Register]

Defines the space between paragraphs.

Effective: next paragraph.

Default: 0.3 v (1 v on low-resolution devices).

`\n[QI]` [Register]

Defines the indentation amount used on both sides of a paragraph set with the `QP` or between the `QS` and `QE` macros.

Effective: next paragraph.

Default: 5 n.

`\n[PORPHANS]` [Register]

Defines the minimum number of initial lines of any paragraph that must be kept together to avoid isolated lines at the bottom of a page. If a new paragraph starts close to the bottom of a page, and there is insufficient space to accommodate `PORPHANS` lines before an automatic page break, `groff ms` forces a page break before formatting the paragraph.[2] This is a GNU extension.

Effective: next paragraph.

Default: 1.

Heading settings

`\n[PSINCR]` [Register]

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

Effective: next heading.

Default: 1 p.

`\n[GROWPS]` [Register]

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

Effective: next heading.

Default: 0.

`\n[HORPHANS]` [Register]

Defines the minimum number of lines of the subsequent paragraph that must be kept on the same page as a section heading. If insufficient space remains on the current page to accommodate the heading and this number of lines of paragraph text, `groff ms` forces a page break before setting the heading.[3]

Effective: next paragraph.

Default: 1.

---

[2] Strictly, `groff ms` requires the equivalent vertical distance that would be required by `\n[PORPHANS]` lines of text set at the type size and vertical spacing configured at the time the document calls the paragraphing macro, plus one vertical motion quantum.

[3] Strictly, `groff ms` requires the equivalent vertical distance that would be required by the heading itself plus the inter-paragraph distance `\n[PD]` and `\n[HORPHANS` lines of text set at the type size and vertical spacing configured at the time the document calls the heading macro, plus one vertical motion quantum.

`\*[SN-STYLE]` [String]

Defines the style used to print numbered headings. See Section 4.6.5.4 [Headings in ms], page 42. This is a GNU extension.

Effective: next heading.

Default: alias of `SN-DOT`

Footnote settings

`\n[FI]` [Register]

Defines the footnote indentation. This is a Berkeley extension.

Effective: next footnote.

Default: 2 n.

`\n[FF]` [Register]

Defines the format of automatically numbered footnotes, and those for which the `FS` request is given a *mark* argument, at the bottom of a column or page. This is a Berkeley extension.

- `0` Set an automatic number[4] as a superscript (on typesetters) or surrounded by square brackets (on terminals). The footnote paragraph is indented as with `PP` if there is an `FS` argument or an automatic number, and as with `LP` otherwise. This is the default.
- `1` As `0`, but set *mark* as regular text, and follow an automatic number with a period.
- `2` As `1`, but without indentation (like `LP`).
- `3` As `1`, but set the footnote paragraph with *mark* hanging (like `IP`).

Effective: next footnote.

Default: 0.

`\n[FPS]` [Register]

Defines the footnote type size.

Effective: next footnote.

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

`\n[FVS]` [Register]

Defines the footnote vertical spacing.

Effective: next footnote.

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

---

[4] defined in Section 4.6.5.10 [ms Footnotes], page 54

`\n[FPD]` [Register]

Defines the footnote paragraph spacing. This is a GNU extension.

Effective: next footnote.

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

`\*[FR]` [String]

Defines the ratio of the footnote line length to the current line length. This is a GNU extension.

Effective: next footnote if single-column layout, next page otherwise.

Default: `11/12`.

Display settings

`\n[DD]` [Register]

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

Effective: next display boundary.

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

`\n[DI]` [Register]

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

Effective: next indented display.

Default: 0.5 i.

Other settings

`\n[MINGW]` [Register]

Defines the default minimum width between columns in a multi-column document. This is a GNU extension.

Effective: next page.

Default: 2 n.

`\n[TC-MARGIN]` [Register]

Defines the width of the field in which page numbers are set in a table of contents entry; the right margin thus moves inboard by this amount. This is a GNU extension.

Effective: next `PX` call.

Default: `\w'000'`

4.6.4 Document Description Macros

Only the simplest document lacks a title.[5] As its level of sophistication (or complexity) increases, it tends to acquire a date of revision, explicitly identified authors, sponsoring institutions for authors, and, at the rarefied heights, an abstract of its content. By default, `ms` .I ms arranges most of the document description (the title, author names and institutions, and abstract, but not the date) at the top of the first page.

Define these data by calling the macros below in the order shown; `DA` or `ND` can be called to set the document date (or other identifier) at any time before (a) the abstract, if present, or (b) its information is required in a header or footer. Use of these macros is optional, except that `TL` is mandatory if any of `RP`, `AU`, `AI`, or `AB` is called, and `AE` is mandatory if `AB` is called.

`.RP` [`no-repeat-info`] [`no-renumber`] [Macro]

Use the "report" (AT&T: "released paper") format for your document, creating a separate cover page. A `no-repeat-info` argument causes `ms` to produce a cover page without repeating any of its information subsequently (but see the `DA` macro below regarding the date). Normally, `RP` sets the page number following the cover page to 1. Specifying the `no-renumber` argument suppresses this alteration. Arguments can occur in any order. `ms` recognizes `no` as a synonym of `no-repeat-info` to maintain AT&T compatibility. Options other than `no` are GNU extensions.

`.TL` [Macro]

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

`.AU` [Macro]

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

`.AI` [Macro]

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

`.DA` [*x* . . .] [Macro]

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

---

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

`.ND` [*x* . . .] [Macro]

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

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

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

`.AE` [Macro]

End the abstract.

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

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

. . .the rest of the paper. . .

4.6.5 Body Text

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

4.6.5.1 Text settings

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

Which font families are available depends on the output device; as a convention, `T` selects a serif family ("Times"), `H` a sans-serif family ("Helvetica"), and `C` a monospaced family ("Courier"). The man page for the output driver documents its font repertoire. Consult the *groff*(1) man page for lists of available output devices and their drivers.

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

4.6.5.2 Typographical symbols

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

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

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

4.6.5.3 Paragraphs

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

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

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

`.IP` [*mark* [*width*]] [Macro]

Set a paragraph with a left indentation. *mark* is not indented and is empty by default. It has several applications; see Section 4.6.5.6 [Lists in ms], page 47. *width* overrides the indentation amount stored in the `PI` register; its default unit is '`n`'. Once specified, *width* applies to further `IP` calls until specified again or a heading or different paragraphing macro is called.

`.QP` [Macro]

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

`.QS` [Macro]
`.QE` [Macro]

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

`.XP` [Macro]

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

The following example illustrates the use of paragraphing macros.

```
.NH 2
Cases used in the 2001 study
.LP
Two software releases were considered for this report.
.PP
The first is commercial software;
the second is free.
.IP \[bu]
Microsoft Word for Windows,
starting with version 1.0 through the current version
(Word 2000).
.IP \[bu]
GNU Emacs,
from its first appearance as a standalone editor through
the current version (v20).
See [Bloggs 2002] for details.
.QP
Franklin's Law applied to software:
software expands to outgrow both RAM and disk space over
time.
.SH
Bibliography
.XP
Bloggs, Joseph R.,
.I "Everyone's a Critic" ,
Underground Press, March 2002.
A definitive work that answers all questions and
criticisms about the quality and usability of free
software.
```

4.6.5.4 Headings

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

`.NH` [*depth*] [*name*] [Macro]
`.NH S` *heading-depth-index* . . . [Macro]

Set an automatically numbered heading.

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

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

The *name* is used to assign a named destination for use with the XR command.

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

An example may be illustrative.

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

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

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

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

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

You can control the style used to print numbered headings by defining an appropriate alias for the string SN-STYLE. By default, SN-STYLE is aliased

to SN-DOT. If you prefer to omit the terminating period from numbers appearing in numbered headings, you may define the alias as follows.

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

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

.SH [*depth*] [*name*] [Macro]

Set an unnumbered heading.

depth is a GNU extension indicating the heading depth corresponding to the *depth* argument of NH. It matches the type size at which the heading is set to that of a numbered heading at the same depth when the GROWPS and PSINCR heading size adjustment mechanism is in effect. The *name* is used to assign a named destination for use with the XR command.

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

The input

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

causes “1. Carnivora” to be printed in 13-point type, followed by “1.1. Felinae” in 11.5-point type, while “1.1.1. Felis catus” and all more deeply nested heading levels remains in the 10-point type specified by the PS register. “Machairodontinae” is printed at 11.5 points, since it corresponds to heading level 2.

In groff ms, the NH and SH macros consult the HORPHANS register to prevent the output of isolated headings at the bottom of a page; it specifies the minimum number of lines of an immediately subsequent paragraph that must be kept on the same page as the heading. If insufficient space remains on the current page to accommodate the heading and this number of lines

of paragraph text, `groff ms` forces a page break before setting the heading. Any display macro call or `tbl`, `pic`, or `eqn` region between the heading and the subsequent paragraph suppresses this grouping. See Section 4.6.5.8 [ms keeps and displays], page 50, and Section 4.6.5.9 [ms Insertions], page 52.

4.6.5.5 Typeface and decoration

The `ms` macros provide a variety of ways to style text. Attend closely to the ordering of arguments labeled *pre* and *post*, which is not intuitive. Support for *pre* arguments is a GNU extension.[6]

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

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

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

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

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

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

`.UL` [*text* [*post*]] [Macro]
Typeset *text* with an underline. On terminals, *text* is bracketed with underscores '_'. *post*, if present, is set after *text* with no intervening space.

---

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

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

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

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

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

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

The foregoing example produces output as follows.

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

You can use the output line continuation escape sequence `\c` to achieve the same result (see Section 5.16 [Line Continuation], page 134). It is also portable to older `ms` implementations.

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

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

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

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

Rather than calling the `CW` macro, in `groff ms` you might prefer to change the font family to Courier by setting the `FAM` string to '`C`'. You can then use all four style macros above, returning to the default family (Times) with '`.ds FAM T`'. Because changes to `FAM` take effect only at the next paragraph,

`CW` remains useful to "inline" a change to the font family, similarly to the practice of this document in noting syntactical elements of `ms` and `groff`.

4.6.5.6 Lists

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

The following is an example of a bulleted list.

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

```
A bulleted list:

• lawyers

• guns

• money
```

The following is an example of a numbered list.

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

```
A numbered list:

1. lawyers

2. guns

3. money
```

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

The next example illustrates a glossary-style list.

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

```
A glossary-style list:

lawyers
      Two or more attorneys.

guns  Firearms, preferably large-caliber.

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

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

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

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

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

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

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

In each case, the result is the same.

```
A glossary-style list:

lawyers
      Two or more attorneys.

guns
      Firearms, preferably large-caliber.

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

4.6.5.7 Indented regions

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

`.RS` [Macro]

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

`.RE` [Macro]

End the (next) most recent indented region.

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

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

```
• Lawyers:

  • Dewey,

  • Cheatham, and

  • Howe.

• Guns
```

4.6.5.8 Keeps, boxed keeps, and displays

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

You can alternatively specify a *floating keep*: if a keep cannot fit on the current page, `ms` holds it, allowing material following the keep (in the source document) to fill the remainder of the current page. When the page breaks by reaching its bottom or by `bp` request, `ms` puts the floating keep at the beginning of the next page. Use floating keeps to house large graphics or tables that do not need to appear exactly where they occur in the source document.

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

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

As an alternative to the keep mechanism, the `ne` request forces a page break if there is not at least the amount of vertical space specified in its argument remaining on the page (see Section 5.18 [Page Control], page 137). One application of `ne` is to reserve space on the page for a figure or illustration to be included later.

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

`.B1` [Macro]
`.B2` [Macro]

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

Boxed keep macros cause breaks; to box words within a line, recall `BX` in Section 4.6.5.5 [Typeface and decoration], page 45. `ms` draws box lines close to the text they enclose so that they are usable within paragraphs. When boxing entire paragraphs thus, you may improve their appearance by calling `B1` after the first paragraphing macro, and invoking the `sp` request before calling `B2`.

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

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

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

`.DS L` [Macro]
`.LD` [Macro]

Begin (`DS`: kept) left-aligned display.

`.DS` [`I` [*indent*]] [Macro]
`.ID` [*indent*] [Macro]

Begin (`DS`: kept) display indented by *indent* if specified, and by the amount of the `DI` register otherwise.

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

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

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

Begin a (`DS`: kept) centered display: each line in the display is centered.

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

Begin a (`DS`: kept) right-aligned display. This is a GNU extension.

`.DE` [Macro]

End any display.

`groff ms` inserts the distance stored in the `DD` register before and after each pair of display macros; this is a Berkeley extension. This distance replaces any adjacent inter-paragraph distance or subsequent spacing prior to a section heading. The `DI` register is a GNU extension; its value is an indentation applied to displays created with '`.DS`' and '`.ID`' without arguments, to '`.DS I`' without an indentation argument, and to indented equations set with '`.EQ`'. Changes to either register take effect at the next display boundary.

The display distance applies even in footnotes (discussed below), which may cause a footnote with a display at its end to "emptily" spill to the next page. Consider the following tactic to compensate.

```
.FS
Recall the ideal gas law.
.nr DD-saved \n[DD] \" stash display distance
.nr DD 0       \" eliminate automatic space around display
.sp \n[DD-saved]u   \" manually put space before it
.EQ
P V = n R T
.EN
.FE
.nr DD \n[DD-saved] \" restore previous setting
```

4.6.5.9 Tables, figures, equations, and references

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

`.TS` [`H`] [Macro]
`.TE` [Macro]

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

`.PS` *h v* [Macro]
`.PE` [Macro]
`.PF` [Macro]

`PS` begins a picture to be processed by the `pic` preprocessor; either of `PE` or `PF` ends it, the latter with "flyback" to the vertical position at its top. Create `pic` input manually or with a program such as `xfig`. *h* and *v* are the horizontal and vertical dimensions of the picture; `pic` supplies them automatically.

`.EQ` [*align* [*label*]] [Macro]
`.EN` [Macro]

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

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

Demarcate a bibliographic citation to be processed by the `refer` preprocessor. *refer*(1) provides a comprehensive reference to the preprocessor and the format of its bibliographic database.

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

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

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

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

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

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

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

Create diagrams with `pic`.

```
.PS
circle "input";
arrow;
box width 1.5i "\f[CR]groff -Rept -ms\f[]";
arrow;
circle "output";
.PE
```

`groff` options run preprocessors on the input: `-e` for `eqn`, `-p` for `pic`, `-R` for `refer`, and `-t` for `tbl`.

4.6.5.10 Footnotes

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

`\*[*]` [String]

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

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

`.FS` [*mark*] [Macro]
`.FE` [Macro]

Begin (`FS`) and end (`FE`) a footnote. `FS` calls `FS-MARK` with any supplied *mark* argument, which is then also placed at the beginning of the footnote text. If *mark* is omitted, the next pending automatic number enqueued by interpolation of the `*` string is used, and if none exists, nothing is prefixed.

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

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

Footnotes can be safely used within keeps and displays, but you should avoid using automatically numbered footnotes within floating keeps. You can

place a second `\**` interpolation between a `\**` and its corresponding `FS` call as long as each `FS` call occurs *after* the corresponding `\**` and occurrences of `FS` are in the same order as corresponding occurrences of `\**`.

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

The `FF` register controls the formatting of automatically numbered footnote paragraphs, and those for which `FS` is given a *mark* argument, See Section 4.6.3 [ms Document Control Settings], page 32.

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,[7] an `FL` register was used for the line length in footnotes; however, setting this register at document initialization time had no effect on the footnote line length in multi-column arrangements.[8]

Prefer the `FR` string over the `FL` register in contemporary documents. The footnote line length is effectively computed as '*`column-width`* `* \*[FR]`'. If you require an absolute footnote line length, recall that `roff` formatters evaluate numeric expressions strictly from left to right, without operator precedence (parentheses are honored).

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

4.6.5.11 Language and localization

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

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

The following strings are available.

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

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

`\*[TOC]` [String]
Contains the string printed at the beginning of the table of contents. The default is '`Table of Contents`'.

---

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

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

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

Contain the full names of the calendar months. The defaults are in English: '`January`', '`February`', and so on.

4.6.6 Page layout

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

4.6.6.1 Headers and footers

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

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

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

The `OH` and `EH` macros define headers for odd- (recto) and even-numbered (verso) pages, respectively; the `OF` and `EF` macros define footers for them.

With either method, a percent sign `%` in header or footer text is replaced by the current page number. By default, `ms` places no header on a page numbered "1" (regardless of its number format).

`.P1` [Macro]

Typeset the header even on page 1. To be effective, this macro must be called before the header trap is sprung on any page numbered "1";

in practice, unless your page numbering is unusual, this means that you should call it early, before TL or any heading or paragraphing macro. This is a Berkeley extension.

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

4.6.6.2 Tab stops

Use the ta request to define tab stops as needed. See Section 5.12 [Tabs and Fields], page 125.

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

4.6.6.3 Margins

Control margins using the registers summarized in "Margin settings" in Section 4.6.3 [ms Document Control Settings], page 32, above. There is no setting for the right margin; the combination of page offset \n[PO] and line length \n[LL] determines it.

4.6.6.4 Multiple columns

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

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

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

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

4.6.6.5 Creating a table of contents

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

Define an entry to appear in the table of contents by bracketing its text between calls to the `XS` and `XE` macros. A typical application is to call them immediately after `NH` or `SH` and repeat the heading text within them. The `XA` macro, used within '`.XS`'/'`.XE`' pairs, supplements an entry—for instance, when it requires multiple output lines, whether because a heading is too long to fit or because style dictates that page numbers not be repeated. You may wish to indent the text thus wrapped to correspond to its heading depth; this can be done in the entry text by prefixing it with tabs or horizontal motion escape sequences, or by providing a second argument to the `XA` macro. `XS` and `XA` automatically associate the page number where they are called with the text following them, but they accept arguments to override this behavior. At the end of the document, call `TC` or `PX` to emit the table of contents; `TC` resets the page number to '`i`' (Roman numeral one), and then calls `PX`. All of these macros are Berkeley extensions.

`.XS` [*page-number*] [Macro]
`.XA` [*page-number* [*indentation*]] [Macro]
`.XE` [Macro]

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

`.PX` [`no`] [Macro]

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

`.TC` [`no`] [Macro]

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

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

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

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

`.XN` *heading-text* [Macro]
`.XH` *depth heading-text* [Macro]

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

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

`.XN-REPLACEMENT` *heading-text* [Macro]
`.XH-REPLACEMENT` *depth heading-text* [Macro]

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

`.XN-INIT` [Macro]
`.XH-UPDATE-TOC` *depth heading-text* [Macro]

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

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

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

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

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

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

4.6.7 Using PDF features with ms

If you run an ms document with `-Tpdf` and `-rPDFFEAT=1` the PDF features shown in this document will be incorporated. The PDF produced will show an overview bookmark panel, the pdf "properties" information will include the document title and author(s), any TOC included will have clickable entries which can navigate the document, and if the document has a cover page (introduced with `RP`) the TOC will be relocated to after the cover.

There are also two new macro calls available.

`.XR` *name* [*post* [*pre* [*hotlink-text*]]] [Macro]

The *name* parameter should match a *name* given to a `NH` or `SH` section/subsection, See Section 4.6.5.4 [Headings in ms], page 42, or a name set by a `XM` command. The *hotlink-text* is the text which will be clickable. If the text ends with "`+`" or "`*`" (the expansion characters), the text associated with the section heading will replace the character (enclosed with double quotes if the final character is "`+`"). If no *hotlink-text* is given, the

value of `\*[spdf:txt_default]` is used. Which is initially set to "`see: +`". (Note the expansion character). But you can change it.

`.XM` *name* [*text*] [Macro]

Mark a particular position in the document with a *name*. Any text given will be used to replace a trailing expansion character in an `XR` command which references the *name*. It is useful if you wish to name an arbitrary location in the document, so you can setup a link to a picture, diagram or even a single paragraph.

Using `XR` for a "forward" link (where the destination is "later" in the document), will not work properly if the link uses expansion characters and `groff` only sees a single pass of the document. So either use the full text, without a trailing expansion character in the forward link, or use the multi-pass command

```
pdfmom --roff -ms -rPDFFEAT=1 file.ms > file.pdf
```

The command can be shortened to `pdfms -rPDFFEAT=1 ...` if you have created a link to the `pdfmom` command called `pdfms`.

Running an `ms` document with `nroff`, is only a single pass, so the above advice about not using expansion characters with forward links also applies.

All of the PDF features mentioned in this document can be disabled if the `groff` command includes `-rPDFFEAT=0`.

4.6.8 Differences from AT&T ms

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

- The internals of `groff ms` differ from those of AT&T and Berkeley `ms`. Documents that depend upon implementation details of those implementations might not format as expected with `groff ms`. It therefore defines the register `GS` .CW GS , which AT&T and Berkeley do not use, as 1. A document can test its value to determine whether it is being formatted with `groff ms` or another implementation.
- The error-handling policy of `groff ms` is to detect and report errors, rather than to ignore them silently.
- Research Tenth Edition Unix supported `P1/P2` macros to bracket code examples; `groff ms` does not.
- `groff ms` does not work in GNU `troff`'s AT&T compatibility mode. If loaded when that mode is enabled, it aborts processing with a diagnostic message.
- Multiple line spacing is not supported. Use a larger vertical spacing instead.

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

  This implies that '\n[PO]' should not be used early in the document, unless it is changed also: accessing an undefined register automatically defines it.
- groff ms supports the PN register, but it is not necessary; you can access the page number via the usual % register and invoke the af request to assign a different format to it if desired.[10]
- The AT&T ms manual documents registers CW and GW as setting the default column width and "intercolumn gap", respectively, and which applied when MC was called with fewer than two arguments. groff

[9] Register values are converted to and stored as basic units. See Section 5.3 [Measurements], page 80.

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

ms instead treats MC without arguments as synonymous with 2C; there is thus no occasion for a default column width register. Further, the MINGW register and the second argument to MC specify a *minimum* space between columns, not the fixed gutter width of AT&T ms.

- The AT&T ms manual did not document the QI register; Berkeley and groff ms do.

\n[GS] [Register]

groff ms sets the register GS to 1; AT&T ms does not use it. A document can test its value to determine whether it is being formatted with groff ms or another implementation.

4.6.8.1 Unix Version 7 ms macros unimplemented by groff ms

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

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

4.6.9 Legacy Features

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

AT&T accent mark strings

AT&T ms defined accent mark strings as follows.

*['] [String]

Apply acute accent to subsequent glyph.

*[`] [String]

Apply grave accent to subsequent glyph.

[11] Removal beforehand is necessary because groff ms aliases these macros with a diagnostic one; you want to reorient the aliased name before (re-)populating the macro.

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

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

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

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

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

Berkeley accent mark and glyph strings

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

4.6.10 Naming Conventions

`groff ms` uses the following conventions for names of macros, strings, and registers. External names available to documents that use the macros contain only uppercase letters and digits. The package reserves the following identifiers for internal use.

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

When selecting a name for your document's own macros, registers, macros, and strings, avoid those reserved by `groff ms` and those defined by GNU `troff`. See Section B.4 [Register Index], page 306, Section B.5 [Macro Index], page 309, and Section B.6 [String Index], page 311, or *groff*(7) for complete lists thereof.

`groff ms` organizes most of its internal names into modules. The naming convenion is 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.
- Names constructed to implement arrays are of the form *array*`!`*index*.

5 GNU troff Reference

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

5.1 Text

AT&T troff was designed to take input as it would be composed on a typewriter, including the teletypewriters used as early computer terminals, and relieve the user drafting a document of concern with details like line length maintenance, hyphenation breaking, and consistent paragraph indentation. Early in its development, the program gained the ability to prepare output for a phototypesetter; a document could then be prepared for output to 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 upon a variety of output formats or devices, including PDF, HTML, laser printers, and terminal displays.

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

5.1.1 Filling

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

GNU troff interprets plain text files employing the Unix line-ending convention. It reads input a character at a time, collecting words as it goes, and fits as many words together on an output line as it can—this is known as *filling*. To GNU troff, a *word* is any sequence of one or more characters that aren't spaces or newlines. The exceptions separate words.[2] To disable filling, see Section 5.9 [Manipulating Filling and Adjustment], page 106.

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

[1] See Section 6.1 [Device and Font Description Files], page 257.

[2] *Tabs* and *leaders* also separate words. *Escape sequences* can function as word characters, word separators, or neither—the last simply have no effect on GNU troff's idea of whether an input character is within a word. We'll discuss all of these in due course.

5.1.2 Sentences

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

```
Hello, world!
Welcome to the wide world of groff.
    ⇒ Hello, world!  Welcome to the wide world of groff.
```

GNU `troff` flags certain characters (normally '`!`', '`?`', and '`.`') as potentially ending a sentence.[4] When GNU `troff` encounters one of these *end-of-sentence characters* at the end of an input line, or one of them is followed by two (unescaped) spaces on the same input line, it appends an inter-word space followed by an inter-sentence space in the 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.
```

Observe the two spaces after "T." in the output. "Barnum" doesn't begin a sentence! What to do? Let us meet our first *escape sequence*, a series of input characters that give instructions to GNU `troff` instead of being used to construct output device glyphs.[5] An escape sequence begins with the backslash character `\` by default, an uncommon character in natural language text, and is *always* followed by at least one other character, hence the term "sequence".

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

---

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

[4] TEX treats '`:`' as ending a sentence for spacing purposes. `roff` formatters do not.

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

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

Adding text caused our input to wrap; now, we don't need `\&` after '`T.`' but we do after '`P.`'. Consistent use of the escape sequence ensures that potential sentence boundaries are robust to editing activities. Further advice along these lines follows in Section 5.1.11 [Input Conventions], page 76.

Normally, the occurrence of a visible non-end-of-sentence character (as opposed to a space or tab) immediately after an end-of-sentence character cancels detection of the end of a sentence. For example, the formatter would be incorrect to infer the end of a sentence after the dot in '`3.14159`'. However, it treats several characters *transparently* after the occurrence of an end-of-sentence character—it does not cancel end-of-sentence status upon encountering them. Such characters are often used as footnote marks or to close quotations and parentheticals. The default set is '`"`', '`'`', '`)`', '`]`', '`*`', `\[dg]`, `\[dd]`, `\[rq]`, and `\[cq]`. The last four are examples of *special characters*, escape sequences whose purpose is to obtain glyphs that are not easily typed at the keyboard, or which have special meaning to the formatter (like `\` itself).[6]

```
\[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)
```

Configure the sets of characters that potentially end sentences or are transparent to sentence endings with the `cflags` request (see Section 5.19.4 [Characters and Glyphs], page 146). Use the `ss` request to change—or eliminate—supplemental inter-sentence space (see Section 5.9 [Manipulating Filling and Adjustment], page 106).

5.1.3 Hyphenation

When an output line is nearly full, it is uncommon for the next word collected from the input to exactly fill it—often, there is room left over for only part of the next word. *Hyphenation* is the process of splitting a word so that it appears partially on one line, followed by a hyphen to indicate to the reader that the word has been broken, and that its remainder lies on the next. Hyphenation break points can be manually specified; GNU `troff` also uses a hyphenation algorithm and language-specific pattern files (based on TEX's) to decide which words can be hyphenated and where.

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

Hyphenation does not always occur even when the hyphenation rules for a word allow it; it can be disabled, and when not disabled there are several parameters that can prevent it in certain circumstances. See Section 5.10 [Manipulating Hyphenation], page 114.

5.1.4 Breaking

Once an output line is full, the formatter places the next word (or remainder of a hyphenated one) on a different output line; this is called a *break*. In this manual and in `roff` discussions generally, a "break" if not further qualified always refers to the termination of an output line. When the formatter is filling text, it introduces breaks automatically to keep output lines from exceeding the configured line length. After an automatic break, the formatter adjusts the line if applicable (see below), and then resumes collecting and filling text on the next output line.

Sometimes, a line cannot be broken automatically. This usually does not happen with natural language text unless the output line length has been manipulated to be extremely short, but it can with specialized text like program source code. We can use `perl` at the shell prompt to contrive an example of failure to break the line. We also employ the `-z` option to suppress normal output.

```
$ perl -e 'print "#" x 80, "\n";' | nroff -z
     error  cannot adjust line; overset by 15n
```

The remedy for these cases is to tell GNU `troff` where the line may be broken without hyphens. This is done with the non-printing break point escape sequence '`\:`'; see Section 5.10 [Manipulating Hyphenation], page 114.

What if the document author wants to stop filling lines temporarily, for instance to start a new paragraph? There are several solutions. A blank input line not only causes a break, but by default it also outputs a one-line vertical space (effectively a blank output line). This behavior can be modified; see Section 5.29.3 [Blank Line Traps], page 214. Macro packages may discourage or disable the blank line method of paragraphing in favor of their own macros.

A line that begins with one or more spaces causes a break. The spaces are output at the beginning of the next line without being *adjusted* (see below); however, this behavior can be modified (see Section 5.29.4 [Leading Space Traps], page 214). Again, macro packages may provide other methods of producing indented paragraphs. Trailing spaces on text lines are discarded.[7]

What if the file ends before enough words have been collected to fill an output line? Or the output line is exactly full but not yet broken, and there is no more input? The formatter breaks the pending output line without adjustment upon encountering the end of input. Certain requests also cause breaks, implicitly or explicitly. This is discussed in Section 5.9 [Manipulating Filling and Adjustment], page 106.

---

[7] See [text lines], page 72. AT&T `troff` also cancels end-of-sentence detection.

5.1.5 Adjustment

After performing an automatic break, the formatter may then *adjust* the line, widening inter-word spaces until the text reaches the right margin. Extra spaces between words are preserved. Leading and trailing spaces are handled as noted above. You can align text to the left or right margin only, or center it; see Section 5.9 [Manipulating Filling and Adjustment], page 106.

5.1.6 Tabs and Leaders

The formatter translates input horizontal tab characters ("tabs") and `Control+A` characters ("leaders") into movements to the next tab stop. Tabs simply move to the next tab stop; leaders place enough periods to fill the space. Tab stops are by default located every half inch measured from the drawing position corresponding to the beginning of the input line; see Section 5.2 [Page Geometry], page 79. Tabs and leaders do not cause breaks and therefore do not interrupt filling. Below, we use arrows → and bullets • to indicate input tabs and leaders, respectively.

```
A→B→C•D→•E
.br \" This is a "request".  See explanation below.
1
→2→3•4
→•5
    ⇒ A       B       C.......D       ........E
    ⇒ 1         2       3.......4         ........5
```

Tabs and leaders lend themselves to table construction.[8] The tab and leader fill characters can be configured, and further facilities for sophisticated table composition are available; see Section 5.12 [Tabs and Fields], page 125. There are many details to track when using such low-level features, so most users turn to the *tbl*(1) preprocessor to lay out tables.

5.1.7 Requests and Macros

The foregoing describes almost all of the syntax of the `roff` language,[9] with an exception already noted in passing. A *request* is an instruction to the formatter that occurs after a *control character*, which is recognized at the beginning of an input line. The regular control character is a dot (`.`). Its counterpart, the *no-break control character*, a neutral apostrophe (`'`), suppresses the break that some requests imply. These characters were chosen because it is uncommon for lines of text in natural languages to begin with them. If you require a formatted period or apostrophe (closing single quotation mark) where the formatter expects a control character, prefix the dot or neutral apostrophe with the dummy character escape sequence, '`\&`'.

---

[8] "Tab" abbreviates "tabulation", suggesting a table arrangement mechanism.

[9] The backspace character is also meaningful. See Section 5.25 [Page Motions], page 191.

An input line beginning with a control character is called a *control line*. Every line of input that is not a control line is a *text line*.[10]

Requests often take *arguments*, words (separated from the request name and each other by spaces) that specify details of the action you expect the formatter to perform.

Requests and escape sequences comprise the control language of the formatter. Of key importance are the requests that define macros. Macros are invoked like requests, permitting extension or override of the request repertoire.[11]

Think of a *macro* as an abbreviation you can define for a collection of control and text lines. When a document *calls* a macro by placing its name after a control character, the formatter replaces the control line with the macro's definition. The process of textual replacement is known as *interpolation*.[12] The formatter performs an interpolation upon recognition, and once performed, it scans the replacement for further requests, macro calls, and escape sequences.

In `roff` systems, the `de` request defines a macro.[13]

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

The foregoing input produces no output by itself; all we have done is store information in a macro named '`DATE`'. Observe the pair of dots that ends the macro definition. This is a default; you can specify your own terminator for the macro definition as the second argument to the `de` request.

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

In fact, the ending mark is itself the name of a macro to be called, or a request to be invoked, if it is defined at the time its control line is read.

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

[10] The `\RET` escape sequence can alter how an input line is classified; see Section 5.16 [Line Continuation], page 134.

[11] Argument handling in macros is more flexible but also more complex. See Section 5.6.3 [Calling Macros], page 92.

[12] Some escape sequences undergo interpolation as well.

[13] GNU `troff` offers additional ones. See Section 5.24 [Writing Macros], page 182.

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

The advantage in typing and maintenance economy may not be obvious from such a short example, but imagine a much longer document with dozens of such paragraphs, each requiring a notice of managerial approval. Consider what must happen if you are in charge of generating a new version of such

a document with a different date, for a different boss. With well-chosen macros, you only have to change each datum in one place.

In practice, we would probably use strings (see Section 5.22 [Strings], page 170) 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 our `DATE` and `BOSS` macros in the opposite order; perhaps less obviously, we could also have defined them *after* `NOTICE`. Such "forward references" are well-defined because the body of a macro definition is, for the most part, stored rather than interpreted (see Section 5.24.2 [Copy Mode], page 188). While a macro is being defined (or appended to), requests are not interpreted and macros not interpolated; some commonly used escape sequences *are* however interpreted. `roff` systems also support recursive macro calls, as long as you have a way to break the recursion (see Section 5.23 [Conditionals and Loops], page 175). Maintainable `roff` documents tend to arrange macro definitions to minimize forward references.

5.1.8 Macro Packages

Macro definitions can be collected into *macro files*, `roff` input files designed to produce no output themselves but instead ease the preparation of other `roff` documents. There is no syntactical difference between a macro file and any other `roff` document; only its purpose distinguishes it. When a macro file is installed at a standard location and suitable for use by a general audience, it is often termed a *macro package*.[14] Macro packages can be loaded by supplying the `-m` option to GNU `troff` or a `groff` front end. Alternatively, a document requiring a macro package can load it with the `mso` ("macro source") request.

5.1.9 Input Format

Organize input to GNU `troff` into lines separated by the Unix newline character (`U+000A`), using the character encoding it recognizes: ISO Latin-1 (8859-1). A document encoded in ISO 646:1991 IRV (US-ASCII), or, equivalently, that 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.[15]

Some control characters (from the sets "C0 Controls" and "C1 Controls" as Unicode describes them) are invalid as input characters. GNU `troff` discards them upon reading.[16] It processes a character sequence "foo", followed by an invalid character and then "bar", as "foobar".

---

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

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

[16] It also emits a warning in category '`input`'. See Section 5.38.1 [Warnings], page 245.

Invalid input characters comprise 0x00, 0x0B, 0x0D–0x1F, and 0x80–0x9F.[17] GNU troff uses some of these code points for internal purposes, making non-trivial the extension of the program to accept UTF-8 or other encodings that use characters from these ranges.

5.1.10 Input Encodings

Recall from Section 2.1 [Groff Options], page 7, that the groff command's -k option runs the preconv preprocessor to perform input character encoding conversions to satisfy GNU troff's requirement of a single-byte encoding compatible with ISO 646:1991 IRV (US-ASCII).

Localization influences automatic hyphenation in two distinct but related respects. A macro file specific to a character coding identifies which character codes correspond to letters expected in the language's hyphenation pattern files and sets up case equivalences for those letters. A language's macro file determines which of these letters are equivalent to other letters for hyphenation purposes.

For example, in English, the letter 'ñ' occurs in loan words. The latin1.tmac and latin9.tmac macro files define a hyphenation code for 'ñ' and make 'Ñ' equivalent to it. The English localization file en.tmac furthermore makes 'ñ' equivalent to 'n'. In Spanish (es.tmac), however, 'ñ' and 'n' are *not* equivalent. The language localization file (see Section 5.10 [Manipulating Hyphenation], page 114) loads an appropriate encoding localization file; a document need not do so directly.

koi8-r
: To use KOI8-R, an encoding for the Russian language, either place '.mso koi8-r.tmac' at the very beginning of your document or supply '-m koi8-r' as a command-line argument to groff. The ru.tmac localization file loads koi8-r.tmac automatically.[18]

latin1
: ISO Latin-1 is an encoding for Western European languages. The de.tmac, en.tmac, it.tmac, and sv.tmac localization files load latin1.tmac automatically.

---

[17] Historically, control characters like ASCII STX, ETX, and BEL (Control+B, Control+C, and Control+G, respectively) have been observed in roff documents, particularly in macro packages employing them as delimiters with an output comparison expression to try to avoid collisions with the content of arbitrary user-supplied parameters (see Section 5.23.1 [Operators in Conditionals], page 175). We discourage this expedient; in GNU troff it is unnecessary (outside of compatibility mode) because the program parses delimited arguments at a different input level than their surrounding context. See Section 5.39 [Implementation Differences], page 248.

[18] KOI8-R code points in the range 0x80–0x9F are not valid input to GNU troff; recall Section 5.1.9 [Input Format], page 74. This restriction should be no impediment to practical documents, as these KOI8-R code points do not encode letters, but box-drawing symbols and characters that are better obtained via special character escape sequences; see *groff_char*(7).

`latin2` To use ISO Latin-2, an encoding for Central and Eastern European languages, invoke '`.mso latin2.tmac`' at the beginning of your document or supply '`-m latin2`' as a command-line argument to `groff`. The `cs.tmac` and `pl.tmac` localization files load `latin2.tmac` automatically.

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

`latin9` ISO Latin-9 succeeds Latin-1; it includes a Euro sign and better coverage for French. To use this encoding, invoke '`.mso latin9.tmac`' at the beginning of your document or supply '`-m latin9`' as a command-line argument to `groff`. The `es.tmac` and `fr.tmac` localization files load `latin9.tmac` automatically.

Some characters from an input encoding may not be available with a particular output driver, or their glyphs may not have representation in the font used. For terminal devices, fallbacks are defined, like '`EUR`' for the Euro sign and '`(C)`' for the copyright sign. For typesetter devices, you may need to "mount" fonts that support glyphs required by the document.[19]

Because a Euro glyph was not historically defined in PostScript fonts, `groff` comes with a font called `freeeuro.pfa` that provides the Euro in several styles. Standard PostScript fonts contain the glyphs from Latin-5 and Latin-9 that Latin-1 lacks, so these encodings are supported for the `ps` and `pdf` output devices as `groff` ships, while Latin-2 is not.

Unicode supports characters from all other input encodings; the `utf8` output driver for terminals therefore does as well. The DVI output driver supports the Latin-2 and Latin-9 encodings if the command-line option '`-m ec`' is used as well.[20]

5.1.11 Input Conventions

Since a `roff` formatter fills text automatically, its experienced users tend to avoid visual composition of text in input files: the esthetic appeal of the formatted output is what matters. Therefore, `roff` input should be arranged such that it is easy for authors and maintainers to compose and develop the document, understand the syntax of `roff` requests, macro calls, and preprocessor languages used, and predict the behavior of the formatter. Several traditions have accrued in service of these goals.

- Follow sentence endings in the input with newlines to ease their recognition; recall Section 5.1.2 [Sentences], page 68. It is frequently convenient to end text lines after colons and semicolons as well, as these typically

[19] See Section 5.19.3 [Font Mounting Positions], page 145.

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

precede independent clauses. Consider doing so after commas; they often occur in lists that become easy to scan when itemized by line, or constitute supplements to the sentence that are added, deleted, or updated to clarify it. Parenthetical and quoted phrases are also good candidates for placement on text lines by themselves.

- Set your text editor's line length to 72 characters or fewer.[21] This limit, combined with the previous item of advice, makes it less common that an input line will wrap in your text editor, and thus will help you perceive excessively long constructions in your text. Recall that natural languages originate in speech, not writing, and that punctuation is correlated with pauses for breathing and changes in prosody.
- Use `\&` after '`!`', '`?`', and '`.`' if they are followed by space or newline characters and don't end a sentence.
- In filled text lines, use `\&` before '`.`' and '`'`' if they are preceded by space, so that revisions to the input don't turn them into control lines.
- Do not use spaces to perform indentation or align columns of a table. Leading spaces are reliable when text is not being filled. (Exception: when laying out a table with GNU `tbl`, specifying the `nospaces` region option causes the program to ignore spaces at the boundaries of table cells.)
- 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 the formatter to ignore the remainder of the input line.
- Use the empty request—a control character followed immediately by a newline—to visually manage separation of material in input files. Many of the `groff` project's own documents use an empty request between sentences, after macro definitions, and where a break is expected, and two empty requests between paragraphs or other requests or macro calls that will introduce vertical space into the document.

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

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

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

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

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

5.2 Page Geometry

roff systems format text under certain assumptions about the size of the output medium, or page. For the formatter to correctly break a line it is filling, it must know the line length, which it derives from the page width.[22] For it to decide whether to write an output line to the current page or wait until the next one, it must know the page length.[23]

A device's *resolution* converts practical units like inches or centimeters to *basic units*, a length measure convenient for the output device or file format. The formatter and output driver use basic units to reckon page measurements. The device description file defines its resolution and page dimensions.[24]

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

Text is arranged on a one-dimensional lattice of text baselines from the top to the bottom of the page. A *text baseline* is a (usually invisible) line upon which the glyphs of a typeface are aligned. *Vertical spacing* is the distance between adjacent text baselines. Typographic tradition sets this quantity to 120% of the type size. Typographers term this unit a *vee*.

While the formatter (and, later, output driver) is processing a page, it keeps track of its *drawing position*, which is the location at which the next glyph will be written, from which the next motion will be measured, or where a geometric object will commence rendering. Notionally, glyphs are drawn from the text baseline upward and to the right.[25] A glyph therefore "starts" at its bottom-left corner. The formatter's origin is one vee below the page top to prevent a glyph from lying partially or wholly off the page.

Further, it is conventional not to write or draw at the extreme edges of the page. Typesetters configure a *page offset*, a rightward shift from the left edge that defines the zero point from which the formatter reckons the line indentation and length.[26]

Combining the foregoing facts results in an origin that lies at the page offset in the horizontal dimension and at the text baseline (using the default vertical spacing) in the vertical dimension. A document can change these prior to its first written or drawn output.[27]

---

[22] See Section 5.15 [Line Layout], page 131.

[23] See Section 5.17 [Page Layout], page 136.

[24] See Section 6.1.1 [DESC File Format], page 257.

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

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

[27] See Section 5.15 [Line Layout], page 131, and Section 5.20 [Manipulating Type Size and Vertical Spacing], page 163.

Vertical spacing has an impact on page-breaking decisions. Generally, when a break occurs, the formatter automatically moves the drawing position to the next text baseline. If the formatter were already writing to the last line that fits on the page, advancing by one vee would place the next text baseline off the page. To avoid that, `roff` formatters instruct the output driver to eject the page, start a new one, and again place the drawing position at the page offset one vee below the page top; this is a *page break*.

When the last line of input text corresponds to the last output line that fits on the page, the break caused by the end of input also breaks the page, producing a useless blank one. Macro packages keep users from having to confront this difficulty by setting "traps";[28] moreover, all but the simplest page layouts tend to have headers and footers, or bear vertical margins of at least one vee.

5.3 Measurements

A `roff` document sometimes requires the input of numeric parameters to specify measurements. Express them as integers or decimal fractions with an optional scaling unit suffixed. A *scaling unit* is a letter that immediately follows the magnitude of a measurement. Digits after the decimal point are optional. Examples of measurements include '`10.5p`', '`11i`', '`.5f`', and '`3.c`'.

The formatter scales measurements by the specified scaling unit, storing them internally (with any fractional part discarded) in basic units. The device resolution can therefore be obtained by storing a value of '`1i`' to a register, then reading the register.

`u` Basic unit; it is at least as small as any other unit.

`i` Inch; defined as 2.54 centimeters.

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

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

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

`s` Scaled point; see Section 5.20.3 [Using Fractional Type Sizes], page 166.

`z` Typographical point; like `p`, but used only with type sizes, to overcome a limitation of AT&T `troff`; see Section 5.20.3 [Using Fractional Type Sizes], page 166.

`f` GNU `troff` defines this unit to scale decimal fractions in the interval [0, 1] to 16-bit unsigned integers. It multiplies a quantity by 65,536. See Section 5.21 [Colors], page 168, for usage.

[28] See Section 5.29 [Traps], page 206.

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

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

`n` En; on typesetters, an en is one-half em, but on terminals an en equals an em, because such devices align all text to a grid of character cells.

`v` Vee; recall Section 5.2 [Page Geometry], page 79.

`M` Hundredth of an em.

5.3.1 Motion Quanta

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

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

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

For example, we might draw short baseline rules on a terminal device as follows. See Section 5.27 [Drawing Geometric Objects], page 200.

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

5.3.2 Default Units

A general-purpose register (one created or updated with the `nr` request)[29] is implicitly dimensionless, or reckoned in basic units if interpreted in a measurement context. But it is convenient for many requests and escape sequences to infer a scaling unit for an argument if none is specified. An explicit scaling unit (not after a closing parenthesis) can override an undesirable default. Effectively, the default unit is suffixed to the expression if a scaling unit is not already present. GNU `troff`'s use of integer arithmetic should also be kept in mind.[30]

---

[29] See Section 5.8 [Registers], page 99.

[30] See Section 5.4 [Numeric Expressions], page 82.

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

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

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

5.4 Numeric Expressions

When evaluated, a *numeric expression* interpolates an integer: it can be as simple as a literal '`0`' or it can be a complex sequence of register and string interpolations interleaved with measurements and operators.

GNU `troff` provides a set of mathematical and logical operators familiar to programmers—as well as some unusual ones—but supports only integer arithmetic.[31] The internal data type used for computing results depends on the host machine but is at least a 32-bit signed integer, which suffices to represent magnitudes within a range of ±2 billion.[32] Arithmetic saturates.[33]

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

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

---

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

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

[33] If overflow would occur, GNU `troff` emits a warning in category '`range`'. See Section 5.38.1 [Warnings], page 245.

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

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

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

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

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

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

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

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

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

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

Caution: Logical operators do not "short-circuit" in `roff` as they do in languages like C. If you want, for instance, to test the value of one register *only* if another has a true value, you can use serial `if` requests.[34]

A logical complementation ("not") operator, `!`, works only within `if`, `ie`, and `while` requests. Further, the formatter recognizes `!` only at the beginning of a numeric expression not contained by another numeric expression. In other words, `!` must be the "outermost" operator. Its presence elsewhere causes the expression to evaluate false.[35] This unfortunate limitation maintains compatibility with AT&T `troff`. Test a numeric expression for falsity within a complex expression by comparing it to a false value.[36]

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

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

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

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

`+` and `-` are also treated differently by the following requests and escape sequences: `bp`, `in`, `ll`, `lt`, `nm`, `nr`, `pl`, `pn`, `po`, `ps`, `pvs`, `rt`, `ti`, `\H`, `\R`, and `\s`. Here, leading plus and minus signs serve as incrementation and decrementation operators, respectively. To negate an expression in these contexts, subtract it from zero or include the unary minus in parentheses with its argument.[37]

A leading `|` operator indicates a measurement relative not to the drawing position but to a boundary. For horizontal motions, the boundary is

---

[34] See Section 5.23 [Conditionals and Loops], page 175.

[35] GNU `troff` emits a warning in category '`number`'. See Section 5.38.1 [Warnings], page 245.

[36] See Section 5.23 [Conditionals and Loops], page 175.

[37] See Section 5.8.1 [Setting Registers], page 99, for examples.

the drawing position corresponding to the beginning of the *input* line. By default, tab stops reckon movements in this way. Most escape sequences do not; | tells them to do so.

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

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

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

In the above example, '|0' specifies a negative motion from the current position (at the end of the argument just emitted, \$1) to the beginning of the input line. Thus, the \l escape sequence in this case draws a line from right to left. A macro call occurs at the beginning of an input line;[38] if the | operator were omitted, then the underline would be drawn at zero distance from the current position, producing device-dependent, and likely undesirable, results. On the 'ps' output device, it underlines the period.

For vertical motions, the | operator specifies a distance from the first text baseline on the page or in the current diversion.[39]

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

In the foregoing example, we've used the \Z escape sequence[40] to restore the drawing position after formatting 'C', then moved vertically to the first text baseline on the page.

---

[38] Control structure syntax creates an exception to this rule, but is designed to remain useful: recalling our example, '.if 1 .Underline this' would underline only "this", precisely. See Section 5.23 [Conditionals and Loops], page 175.

[39] See Section 5.30 [Diversions], page 217.

[40] See Section 5.25 [Page Motions], page 191.

`\B'input'` [Escape sequence]

Interpolate 1 if *input* is a valid numeric expression, and 0 otherwise. The delimiter need not be a neutral apostrophe.[41]

You might use `\B` along with the `if` request to filter out invalid macro or string arguments.[42]

```
.\" Indent by amount given in first argument; assume ens.
.de Indent
.  if \B'\\$1' .in (n;\\$1)
.  el          .tm \\$0: invalid number '\\$1'
..
```

A register interpolated as an operand in a numeric expression must have an Arabic format; luckily, this is the default.[43]

Because spaces separate arguments to requests, spaces are not allowed in numeric expressions unless parentheses surround the (sub)expression containing them.[44]

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

The `nr` request[45] expects its second and optional third arguments to be numeric expressions; a bare `+` does not qualify, so our first attempt elicited an error diagnostic.

5.5 Identifiers

An *identifier* labels a GNU `troff` datum such as a register, name (macro, string, or diversion), typeface (font, family, or style), color, special character or character class, hyphenation language code, environment, or stream. Valid identifiers consist of one or more ordinary characters.[46] An *ordinary*

---

[41] See Section 5.6.5 [Delimiters], page 96.

[42] See Section 5.23 [Conditionals and Loops], page 175.

[43] See Section 5.8.4 [Assigning Register Formats], page 103.

[44] See Section 5.6.2 [Invoking Requests], page 90, and Section 5.23 [Conditionals and Loops], page 175.

[45] See Section 5.8.1 [Setting Registers], page 99.

[46] Use of escape sequences in identifiers is not portable. For example, DWB 3.3 `troff` accepts `\_`. Plan 9 `troff` does too, along with `\'`, `\``, and `\-`. Solaris `troff` rejects all of these except `_`, but accepts `\&`, `\{`, `\}`, `\SPC`, `\%`, and `\c`. Heirloom Doctools `troff` rejects all of these, including `_`, but accepts `\!`, which the others reject. GNU `troff` rejects all of the foregoing.

character is any Unicode Basic Latin character that is not a space and not the escape character; recall Section 5.1.9 [Input Format], page 74. Thus, the identifiers ‘br’, ‘PP’, ‘end-list’, ‘10’, ‘ref*normal-print’, ‘|’, ‘@_’, and ‘!"#$%'()*+,-./’ are all valid. Discretion should be exercised to prevent confusion. Identifiers starting with ‘(’ or ‘[’ require care.

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

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

Beginning a macro, string, or diversion name with the character ‘[’ or ‘]’ forecloses use of the refer preprocessor, which recognizes input lines starting with ‘.[’ and ‘.]’ as bibliographic reference delimiters.

\A'*input*' [Escape sequence]

Interpolate 1 if *input* is a valid identifier, and 0 otherwise. The delimiter need not be a neutral apostrophe.[47] Because GNU troff ignores any input character with an invalid code when reading it, invalid identifiers are empty or contain spaces, tabs, newlines, or escape sequences that interpolate something other than a sequence of ordinary characters.

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

[47] See Section 5.6.5 [Delimiters], page 96.

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

In this example, we also validated the numeric arguments; the registers '`point!x`' and '`point!y`' remain undefined. Recall Section 5.4 [Numeric Expressions], page 82, regarding the `\B` escape sequence.

The formatter's handling of undefined identifiers is context-dependent. There is no way to invoke an undefined request; such syntax is interpreted as a macro call instead. If the identifier is interpreted as a string, macro, or diversion name, the formatter defines it as empty and interpolates nothing.[48] Similarly, if the identifier is interpreted as a register name, the formatter initializes it to zero and interpolates that value.[49] Attempting to use an undefined typeface, special character or character class, color, environment, hyphenation language code, or stream generally provokes an error diagnostic.

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

```
.de xx
foo
..
.
.di xx
bar
.br
.di
.
.xx
    ⇒ bar
```

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

[48] GNU `troff` emits a warning in category '`mac`'. See Section 5.38.1 [Warnings], page 245.
[49] GNU `troff` emits a warning in category '`reg`'. See Section 5.38.1 [Warnings], page 245.

5.6 Formatter Instructions

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

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

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

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

5.6.1 Control Characters

The mechanism of using roff's control characters to invoke requests and call macros was introduced in Section 5.1.7 [Requests and Macros], page 71. The formatter recognizes a control character only at the beginning of an input line, or at the beginning of a branch of a control structure request.[50]

A few requests cause a break implicitly; invoke them with the no-break control character to prevent the break. Break suppression is its sole behavioral distinction. Employing the no-break control character to invoke requests that don't cause breaks is harmless but poor style.[51]

The cc and c2 requests respectively change the control '.' and no-break control '' characters to any desired ordinary character (recall Section 5.5 [Identifiers], page 86), but must be distinct.

.cc [*o*] [Request]

Recognize the ordinary character *o* as the control character. If *o* is absent or invalid, the default control character '.' is selected. If *o* (or '.' if *o* is invalid) is already the escape or no-break control character, an error is diagnosed and the request ignored. The identity of the control character is associated with the environment.[52]

.c2 [*o*] [Request]

Recognize the ordinary character *o* as the no-break control character. If *o* is absent or invalid, the default no-break control character '' is selected. If *o* (or '' if *o* is invalid) is already the escape or control character, an

[50] See Section 5.23 [Conditionals and Loops], page 175.

[51] See Section 5.9 [Manipulating Filling and Adjustment], page 106.

[52] See Section 5.32 [Environments], page 224.

error is diagnosed and the request ignored. The identity of the no-break control character is associated with the environment.[53]

While a macro definition is interpreted, GNU `troff`'s `.br` register permits discovery of the control character used in its call.

`\n[.br]` [Register]

This read-only register interpolates 1 if the currently executing macro was called using the normal control character and 0 otherwise. If a macro is interpolated as a string, the `.br` register inherits its value from the context of the string interpolation.[54]

Use the `.br` register to reliably wrap or intercept requests that imply breaks.

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

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

5.6.2 Invoking Requests

A control character is optionally followed by tabs and/or spaces and then an identifier naming a request or macro. The invocation of an unrecognized request is interpreted as a macro call. Defining a macro with the same name as a request replaces the request. Deleting a request name with the `rm` request makes it unavailable. The `als` request can alias requests, permitting them to be wrapped or non-destructively replaced.[55]

There is no inherent limit on argument length or quantity. Most requests take one or more arguments, and ignore any they do not expect. A request that is not supplied a mandatory argument does nothing.[56] Spaces or tabs separate a request's name from its first argument, but only spaces can separate an argument from its successor.[57] Only one space between arguments is necessary; the formatter ignores any excess. GNU `troff` does not interpret tabs as argument separators.[58]

Generally, a space *within* a request argument is not relevant, is not meaningful, or is supported by bespoke provisions, as with the `tl` request's de-

---

[53] See Section 5.32 [Environments], page 224.

[54] See Section 5.22 [Strings], page 170.

[55] See Section 5.22 [Strings], page 170.

[56] GNU `troff` emits a warning in category '`missing`'. See Section 5.38.1 [Warnings], page 245.

[57] In compatibility mode, a space is not necessary after a request or macro name of two characters' length.

[58] Plan 9 `troff` does.

limiters.[59] Some requests, like ds, interpret the remainder of the control line as a single argument.[60]

Commonly, authors use spaces and/or tabs after a control character to indent the source of documents or macro files. The formatter ignores spaces and tabs immediately after a control character, except that the presence of a tab prevents recognition of the end of a macro definition.

```
.de center
.  if \\n[.br] \
.    br
.  ce \\$1
..
.
.
.de right-align
.→if \\n[.br] \
.→→br
.→rj \\$1
..
```

If you assign an empty blank line trap, you can separate macro definitions (or any input lines) with blank lines.[61]

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

.de center
.  if \\n[.br] \
.    br
.  ce \\$1
..


.de right-align
.→if \\n[.br] \
.→→br
.→rj \\$1
..

.blm             \" deactivate blank line trap
```

[59] See Section 5.17 [Page Layout], page 136.
[60] See Section 5.22 [Strings], page 170.
[61] See Section 5.29.3 [Blank Line Traps], page 214.

5.6.3 Calling Macros

If a macro of the desired name does not exist when called, the formatter creates it and assigns it an empty definition.[62] Calling an undefined macro *does* end a macro definition naming it as its end macro, unless a tab lies between the control character and its name.[63] Because the macro definition is empty, its arguments are discarded.

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

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

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

The first line calls `uh` with three arguments: '`The`', '`Mouse`', and '`Problem`'. The remainder call the `uh` macro with one argument, '`The Mouse Problem`'. The last solution, using escaped spaces, can be found in documents prepared for AT&T `troff`. It can cause surprise when text is adjusted, because `\SPC` inserts a *fixed-width*, non-breaking space. GNU `troff`'s `\~` escape sequence inserts an adjustable, non-breaking space.[64]

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

In GNU `troff`'s AT&T compatibility mode, these characters remain available as `\(rs` and `\(dq`, respectively. AT&T `troff` did not consistently define these special characters, but its descendants can be made to support them.[65]

If even that is not feasible, options remain. To obtain a literal escape character in a macro argument, you can simply type it if you change or disable the escape character first.[66] Otherwise, you must escape the escape character repeatedly to a context-dependent extent.[67]

For the (neutral) double quote, you have recourse to an obscure syntactical feature of AT&T `troff`. Because a double quote can begin a macro argument, the formatter keeps track of whether the current argument was started thus, and doesn't require a space after the double quote that ends it.[68] In the argument list to a macro, a double quote that *isn't* preceded by

---

[62] GNU `troff` emits a warning in category '`mac`'. See Section 5.38.1 [Warnings], page 245.

[63] See Section 5.24 [Writing Macros], page 182.

[64] `\~` is fairly portable; see Section 5.39.3 [Other Differences], page 251.

[65] See Section 6.1 [Device and Font Description Files], page 257.

[66] See Section 5.6.4 [Using Escape Sequences], page 93.

[67] See Section 5.24.2 [Copy Mode], page 188.

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

a space *doesn't* start a macro argument. If not preceded by a double quote that began an argument, this double quote becomes part of the argument. Further, within a quoted argument, a pair of adjacent double quotes becomes a literal double quote.

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

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

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

Outside of compatibility mode,[69] GNU troff doesn't exhibit this problem because it tracks the nesting depth of interpolations.

5.6.4 Using Escape Sequences

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

Escape sequences vary in length. Some take an argument, and of those, some have different syntactical forms for a one-character, two-character, or arbitrary-length argument. Others accept *only* an arbitrary-length argument. In the former scheme, a one-character argument follows the function character immediately, an opening parenthesis '(' introduces a two-character argument (no closing parenthesis is used), and an argument of arbitrary length is enclosed in brackets '[]'. In the latter scheme, the user selects a delimiter character. A few escape sequences are idiosyncratic, and support

[69] See Section 5.39.2 [Compatibility Mode], page 248.

both of the foregoing conventions (\s), designate their own termination sequence (\?), consume input until the next newline (\!, \", \#), or support an additional modifier character (\s again, and \n). In no case can an escape sequence parameter contain an unescaped newline. As with requests, use of some escape sequences in source documents may interact poorly with a macro package you use; consult its documentation to learn of "safe" sequences or alternative facilities it provides to achieve the desired result.

If the character that follows the escape character does not identify a valid operation, the formatter ignores the escape character.[70] In copy mode, the escape character can be used to quote the escape or control characters.[71]

```
$ groff -T ps -ww
.nr N 12
.ds co white
.ds animal elephant
I have \fI\nN \*(co \*[animal]s,\f[]
said \P.\&\~Pseudo Pachyderm.
    error  warning: ignoring escape character before 'P'
    ⇒ I have 12 white elephants, said P. Pseudo Pachyderm.
```

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

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

In the above, the syntax form '\f(' accepts only two characters for an argument; the example works because the subsequent escape sequences are interpolated before the selection escape sequence argument is processed, and strings family and style interpolate one character each.[72]

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

\e [Escape sequence]

Interpolate the escape character. The formatter interprets \e even in copy mode.[73]

---

[70] GNU troff emits a warning in category 'escape'. See Section 5.38.1 [Warnings], page 245.

[71] See Section 5.24.2 [Copy Mode], page 188.

[72] The omission of spaces before the comment escape sequences is necessary; see Section 5.22 [Strings], page 170.

[73] See Section 5.24.2 [Copy Mode], page 188.

The \[rs] special character escape sequence formats a backslash glyph. In macro and string definitions, the input sequences \\ and \E defer interpretation of escape sequences.[74]

.eo [Request]

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

.ec [*o*] [Request]

Recognize the ordinary character *o* as the escape character. If *o* is absent or invalid, the default escape character '\' is selected. If *o* (or '\' if *o* is invalid) is already the control or no-break control character, an error is diagnosed and the request ignored.

Switching escape sequence interpretation off to define a macro and back on afterward can obviate the need to double the escape character within the definition. See Section 5.24 [Writing Macros], page 182. This technique is not available if your macro needs to interpolate values at the time it is *defined*—but many do not.

```
.\" simplified `BR` macro from the man(7) macro package
.eo
.de BR
.  ds result \&
.  while (\n[.$] >= 2) \{\
.    as result \fB\$1\fR\$2\"
.    shift 2
.  \}
.  if \n[.$] .as result \fB\$1\"
\*[result]
.  rm result
.  ft R
..
.ec
```

.ecs [Request]
.ecr [Request]

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

Use these requests together to temporarily change the escape character.

Using a different escape character, or disabling it, when calling macros not under your control will likely cause errors, since GNU troff has no mechanism to "intern" macros—that is, to convert a macro definition into

[74] See Section 5.24.2 [Copy Mode], page 188.

a form independent of its representation.[75] When a macro is called, its contents are interpreted literally.

5.6.5 Delimiters

Some escape sequences that require parameters delimit them. The neutral apostrophe `'` is a popular delimiter choice and shown in this document. The neutral double quote `"` is also commonly seen. Punctuation characters are the best choice (and most portable to other `troff`s), except for those meaningful in numeric expressions; see below.

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

The following escape sequences are not themselves delimited, and thus are allowed as delimiters: `\SPC`, `\%`, `\|`, `\^`, `\{`, `\}`, `\'`, `\``, `\-`, `_`, `\!`, `\?`, `\)`, `\/`, `\,`, `\&`, `\:`, `\~`, `\0`, `\a`, `\c`, `\d`, `\e`, `\E`, `\p`, `\r`, `\t`, and `\u`. However, we discourage using them this way; they can make the input confusing to read.[76] An invalid escape sequence is valid as a delimiter if the character after the escape character would be valid.

The escape sequences `\D`, `\h`, `\H`, `\l`, `\L`, `\N`, `\R`, `\s`, `\S`, `\v`, and `\x` prohibit delimiters that are meaningful in numeric expressions, because they accept numeric expressions as (or within) their arguments. For consistency, GNU `troff` prohibits the same delimiters in the argument to the `tl` request.[77] The `if`, `ie`, and `while` requests each interpret their first argument as a conditional expression;[78] only characters that are not meaningful as operators in that context can be used as output comparison delimiters. The following inputs are therefore invalid as delimiters in GNU `troff`.

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

Delimiter syntax is flexible (and laborious to describe) primarily for historical reasons; the foregoing restrictions need be kept in mind mainly when using GNU `troff` in AT&T compatibility mode. Normally, GNU `troff` keeps track of the nesting depth of escape sequence interpolations, so the only characters you need to avoid using as delimiters are those that appear in the arguments you input, not those that result from interpolation. Typically, `'` works fine.[79]

---

[75] TEX does have such a mechanism.

[76] The GNU *eqn*(1) and *tbl*(1) preprocessors use parameterized but non-delimited special character escape sequences `\(` and `\[` to bracket portions of their output.

[77] See Section 5.17 [Page Layout], page 136.

[78] See Section 5.23.1 [Operators in Conditionals], page 175.

[79] See Section 5.39 [Implementation Differences], page 248.

```
$ groff -T ps
.de Mw
.  nr wd \w'\\$1'
.  tm "\\$1" is \\n(wd units wide.
..
.Mw Wet'suwet'en
.Mw Wet+200i
.cp 1 \" turn on compatibility mode
.Mw Wet'suwet'en
.Mw Wet'
.Mw Wet+200i
     error  "Wet'suwet'en" is 54740 units wide.
     error  "Wet'+200i" is 42610 units wide.
     error  "Wet'suwet'en" is 15860 units wide.
     error  "Wet'" is 15860 units wide.
     error  "Wet'+200i" is 14415860 units wide.
```

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

5.7 Comments

One of the most common forms of escape sequence is the comment.[80]

`\"` [Escape sequence]

Start a comment; read everything up to the next newline in copy mode (see Section 5.24.2 [Copy Mode], page 188) and discard it. The formatter interprets `\"` even in copy mode.[81]

It can be tricky to keep the comments from interfering with the appearance of the output. If the escape sequence is to the right of some text or a request, that portion of the line is ignored, but GNU `troff` processes spaces preceding it normally. This affects requests that read the remainder of the control line as a single argument, including `ds`, `as`, `tm`, and `char`; their variants; as well as `ab`, `device`, `length`, `output`, `pi`, `pso`, `rd`, `sy`, `write`, and `writec`.

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

The formatter handles a `\"` comment on a line by itself as a blank line, because after eliminating the comment, that is all that remains.

---

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

[81] See Section 5.24.2 [Copy Mode], page 188.

```
.nf
apples bananas
\" cantaloupes
durians
    ⇒ apples bananas
    ⇒
    ⇒ durians
```

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

Another commenting scheme sometimes seen is three consecutive neutral apostrophes (`'''`) at the beginning of an input line. This works,[82] but GNU `troff` emits a warning diagnostic (if enabled) about an undefined macro (namely '`'''`').

\# [Escape sequence]

Start a whole-line comment; read everything up to and including the next newline in copy mode[83] and discard it. GNU `troff` introduced this extension to avoid the problems described above. (`\"` is still widely seen, and remains useful for partial-line comments on control lines.) The formatter interprets `\#` even in copy mode.[84]

```
.nr in-indonesia 1
apples bananas \" common favorites
\# cantaloupes
.ie \n[in-indonesia] durians \" Borneo, Sumatra
.el                  elderberries \" England, France
    ⇒ apples bananas durians
```

If we change the comment escape sequence from `\"` to `\#` on the line with the `ie` request, we get the following undesired output.

```
    ⇒ apples bananas durians .el elderberries
```

.ig [*end*] [Request]

Ignore input until, in the current conditional block (if any),[85] the macro *end* is called at the start of a control line, or, if *end* is omitted, the control line '`..`' is encountered. The formatter processes the ignored input as if it were a macro definition, but discards rather than storing it.[86]

---

[82] except in copy mode on Plan 9 `troff`

[83] See Section 5.24.2 [Copy Mode], page 188.

[84] See Section 5.24.2 [Copy Mode], page 188.

[85] See Section 5.23.4 [Conditional Blocks], page 179.

[86] Exception: auto-incrementing registers defined outside the ignored region *will* be modified if interpolated with `\n`± inside it. See Section 5.8.3 [Auto-increment], page 102.

```
.ll 45n
hand\c
.de TX
fasting
..
.ig TX
This is part of a large block of input that has been
temporarily(?) commented out.
.TX
shake
```

⇒ `handfasting shake`

Observe the result if we remove the `ig` request and the call of its end macro.

```
⇒ handThis  is  part  of a large block of input
⇒ that has been temporarily(?)  commented  out.
⇒ shake
```

5.8 Registers

In the `roff` language, numbers and measurements can be stored in *registers*. Many built-in registers exist, supplying anything from components of the date to details of formatting parameters; some of these are read-only. You can also define your own. Recall Section 5.5 [Identifiers], page 86, regarding the construction of valid names for registers. The register name space is separate from that used for requests, macros, strings, and diversions.

Each register (except read-only ones) can be assigned a *format*, causing its value to interpolate with leading zeroes, in Roman numerals, or alphabetically. Some read-only registers are string-valued, meaning that they interpolate text and lack a format.

5.8.1 Setting Registers

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

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

Set register *ident* to *value*. If *ident* doesn't exist, the formatter creates it. In the `\R` escape sequence, the delimiter need not be a neutral apostrophe; recall Section 5.6.5 [Delimiters], page 96.

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

⇒ `1 1`

Later, we will discuss additional forms of `nr` and `\R` that can change a register's value after it is dereferenced but before it is interpolated.[87]

GNU `troff` does not tokenize `\R` when reading it; the escape sequence updates only the formatter's register dictionary and does not contribute (directly) to output.[88]

Further surprise can occur if you use registers like `.k`,[89] whose values are not determined until they are interpolated.

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

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

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

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

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

Increment (decrement) register *ident* by *value*. In the `\R` escape sequence, the delimiter need not be a neutral apostrophe; recall Section 5.6.5 [Delimiters], page 96.

---

[87] See Section 5.8.3 [Auto-increment], page 102.
[88] See Section 5.37 [GNU `troff` Internals], page 237.
[89] See Section 5.25 [Page Motions], page 191.

```
.nr a 1
.nr a +1
\na
    ⇒ 2
```

A `roff` formatter always interprets a leading minus sign in *value* as a decrementation operator, not an algebraic sign. To assign a register a negative value or the negated value of another register, you must force the formatter to interpret '`-`' as a negation or minus, rather than decrementation, operator: enclose the '`-`' with its operand in parentheses or subtract the expression of interest from zero.

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

If *reg* is undefined, the formatter creates it and applies an increment or decrement to it as if to 0.

`.rr` *reg* . . . [Request]

Remove each register *reg*. Technically, only the name is removed; the register's contents are still accessible under aliases created with `aln`, if any. The formatter performs no operation on a nonexistent *reg*. If *reg* is an invalid identifier, GNU `troff` diagnoses an error, but processes any subsequent valid *reg* arguments.

This request is incorrectly documented in the AT&T `troff` manual as accepting only one argument.

`.rnn` *ident1 ident2* [Request]

Rename register *ident1* to *ident2*. Renaming a built-in register does not otherwise alter its properties. If *ident1* is undefined, GNU `troff` ignores the request.[90]

`.aln` *new-register existing-register* [Request]

Create alias (additional name) *new-register* of *existing-register*, causing the names to refer to the same stored object. If *existing-register* is undefined, GNU `troff` ignores the request.[91]

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

[90] GNU `troff` emits a warning in category '`reg`'. See Section 5.38.1 [Warnings], page 245.

[91] GNU `troff` emits a warning in category '`reg`'. See Section 5.38.1 [Warnings], page 245.

5.8.2 Interpolating Registers

The `\n` escape sequence interpolates register contents.

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

Interpolate register with name *ident* (one-character name *i*, two-character name *id*). If the register is undefined, the formatter creates it and assigns it a value of '`0`', and interpolates that value.[92] The formatter interprets `\n` even in copy mode.[93]

```
.nr a 5
.nr as \na+\na
\n(as
    ⇒ 10
.nr a1 5
.nr ab 6
.ds str b
.ds num 1
\n[a\n[num]]
    ⇒ 5
\n[a\*[str]]
    ⇒ 6
```

5.8.3 Auto-increment

User-defined registers can also be incremented or decremented by a configured amount at the time they are interpolated. The value of the increment is specified with a third argument to the `nr` request, and a special interpolation syntax alters and then retrieves the register's value. Together, these features are called *auto-increment*.[94]

`.nr` *ident value incr* [Request]

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

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

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

---

[92] GNU `troff` emits a warning in category '`reg`'. See Section 5.38.1 [Warnings], page 245.
[93] See Section 5.24.2 [Copy Mode], page 188.
[94] A negative auto-increment can be considered an "auto-decrement".

\n+[*ident*] [Escape sequence]
\n-[*ident*] [Escape sequence]

Increment or decrement *ident* (one-character name *i*, two-character name *id*) by the register's auto-incrementation value and then interpolate the new register value. If *ident* has no auto-incrementation value, GNU troff interpolates its value without alteration.

```
.nr a 0 1
.nr xx 0 5
.nr foo 0 -2
\n+a, \n+a, \n+a, \n+a, \n+a
.br
\n-(xx, \n-(xx, \n-(xx, \n-(xx, \n-(xx
.br
\n+[foo], \n+[foo], \n+[foo], \n+[foo], \n+[foo]
    ⇒ 1, 2, 3, 4, 5
    ⇒ -5, -10, -15, -20, -25
    ⇒ -2, -4, -6, -8, -10
```

To change the increment value without changing the value of a register, assign the register's value to itself by interpolating it, and specify the desired increment normally. Apply an increment of '0' to disable auto-incrementation of the register.

5.8.4 Assigning Register Formats

A writable register's value can be interpolated in several number formats. Conventional Arabic numerals are the default format; others see use in sectioning and outlining schemes and alternative page numbering arrangements.

.af *reg fmt* [Request]

Apply number format *fmt* when interpolating register *reg*. The default format for all writable registers is '0'. Specifying an unrecognized format is an error. Valid formats are as follows.

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

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

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

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

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

The formatter interpolates zero values as '0' and prefixes negative quantities with '-' irrespective of format. In Arabic formats, the sign supplements the field width. Assigning a format to a nonexistent register creates it with a zero value.

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

The representable extrema in the 'i' and 'I' formats correspond to Arabic ±3,999.[95]

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

[95] In compatibility mode, GNU troff uses 'w' and 'z' to represent 5,000 and 10,000 in Roman numerals, respectively, following the convention of AT&T troff, and increasing the representable range of register values to Arabic ±39,999. Following Unicode's recommendation, GNU troff outputs Roman numerals using only alphabetic characters, not Roman numeral characters from Unicode's Number Forms block.

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

Interpolate the format of the register *reg* (one-character name *r*, two-character name *rg*). Zeroes represent Arabic formats. If *reg* is not defined, *reg* is not created and nothing is interpolated. The formatter interprets \g even in copy mode.[96]

GNU troff interprets only Arabic numerals. The Roman numeral or alphabetic formats cannot be used as operands to arithmetic operators in expressions; recall Section 5.4 [Numeric Expressions], page 82. For instance, it may be desirable to test the page number independently of its format.

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

5.8.5 Built-in Registers

Predefined registers whose identifiers start with a dot are read-only. Many are Boolean-valued, interpolating a true or false value testable with the if, ie, or while requests.

Caution: Read-only registers are subject to removal like others; once removed, they can be recreated only as normal writable registers and will not otherwise reflect the configuration of the formatter.

A register name is often associated with a request of the same name (without the dot). A complete listing of all built-in registers can be found in Section B.4 [Register Index], page 306.

We present here a few built-in registers that are not described elsewhere in this manual; they have to do with invariant properties of GNU troff, or obtain information about its command-line options or processing progress. All are read-only.

[96] See Section 5.24.2 [Copy Mode], page 188.

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

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

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

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

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

`\n[.R]` Count of available unused registers; in GNU `troff` this register always interpolates the maximum representable integer.[97] Favor its use over numeric literals with many zeroes or nines to indicate an arbitrary large quantity.

`\n[.T]` Indicator of output device selection (Boolean-valued); see `groff`'s `-T` option (Section 2.1 [Groff Options], page 7).

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

`\n[.x]` Major version number of the running GNU `troff` formatter. For example, if the version number is 1.24.2, then `.x` contains '`1`'.

`\n[.y]` Minor version number of the running GNU `troff` formatter. For example, if the version number is 1.24.2, then `.y` contains '`24`'.

`\n[.Y]` Revision number of the running GNU `troff` formatter. For example, if the version number is 1.24.2, then `.Y` contains '`2`'.

5.9 Manipulating Filling and Adjustment

When an output line is pending (see below), a break moves the drawing position to the beginning of the next text baseline, interrupting filling. Recall Section 5.1.4 [Breaking], page 70. The `br` request likewise causes a break. Several other requests imply breaks: `bp`, `brp`, `ce`, `cf`, `fi`, `fl`, `in`, `nf`, `rj`, `sp`, `ti`, and `trf`. If the no-break control character is used with any of these requests, GNU `troff` suppresses the break; instead the requested operation takes effect at the next break. '`''br`' and '`''brp`' do nothing.

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

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

Output line properties like page offset, indentation, adjustment, and even the location of its text baseline, are not determined until the line has been broken. An output line is said to be *pending* if some input has been collected but an output line corresponding to it has not yet been written; such an output line is also termed *partially collected*. If no output line is pending, it is as if a break has already happened; additional breaks, whether explicit or implicit, have no effect. If the vertical drawing position is negative—as it is when the formatter starts up—a break starts a new page (even if no output line is pending) unless an end-of-input macro is being interpreted. See Section 5.29.5 [End-of-input Traps], page 214.

`.br` [Request]

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

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

You can prevent a break between words, as with a quantity and its units.

`\~` [Escape sequence]

Insert an adjustable, unbreakable space. As with ordinary spaces, the formatter discards any sequence of these at the end of an output line if a break occurs.

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

By default, the formatter fills text and adjusts it to reach the output line length. The `nf` request disables filling; the `fi` request reënables it.

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

Enable filling of output lines; a pending output line is broken. The read-only register `.u` is set to 1. The filling enablement status, sometimes

called *fill mode*, is associated with the environment (see Section 5.32 [Environments], page 224). See Section 5.16 [Line Continuation], page 134, for interaction with the `\c` escape sequence.

`.nf` [Request]

Disable filling of output lines: the output line length (see Section 5.15 [Line Layout], page 131) is ignored and output lines are broken where the input lines are. A pending output line is broken and adjustment is suppressed. The read-only register `.u` is set to 0. The filling enablement status is associated with the environment (see Section 5.32 [Environments], page 224). See Section 5.16 [Line Continuation], page 134, for interaction with the `\c` escape sequence.

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

Enable output line adjustment in *mode*, taking effect when the pending (or next) output line automatically breaks. Adjustment is suppressed when filling is, and when a break is directed or implied by another request. (Exceptionally, the `brp` request and `\p` escape sequence break *and* adjust the line.) *mode* can have one of the following values.

`b`
`n` Adjust "normally": if the output line does not consume the distance between the indentation and the configured output line length, GNU `troff` stretches adjustable spaces within the line until that length is reached. When the indentation is zero, this mode spreads the line to both the left and right margins. This is the GNU `troff` default.

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

`l` Align text to the left without adjusting it.

`r` Align text to the right without adjusting it.

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

The adjustment mode and enablement status are encoded in the read-only register `.j`. These parameters are associated with the environment (see Section 5.32 [Environments], page 224).

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

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

`.na` [Request]

Disable output line adjustment, produciing the same output as left-alignment, but altering the value of the adjustment mode register `.j` differently. The adjustment mode and enablement status are associated with the environment.[98]

Normally, an explicit break implies non-adjustment of the pending output line, as at the end of a paragraph.

`.brp` [Request]

`\p` [Escape sequence]

The `brp` request commands a break as `br` does, but also forces adjustment of the output line per the current adjustment mode. Like `br`, it does nothing if invoked with the no-break control character.

`\p` schedules a break with adjustment at the next word boundary. The escape sequence is itself neither a break nor a space of any kind; it can

[98] See Section 5.32 [Environments], page 224.

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

`\p` is typically used for fine-tuning of typeset output late in the document revision process. One of its applications is prevention of a break after an explicit hyphen when this occurs in an undesired place, such as at the end of a recto page, or before a displayed figure. The hyphenation mode can be configured to prevent breaks after *automatically* placed hyphens, but not explicit ones.[99] What one can do in this scenario is place `\p` at the end of the word *before* the one that breaks undesirably.

```
.ll 1.375i
The next data were out-of-band. \" breaks after "out-"
.br
The next data were\p out-of-band. \" breaks after "were"
```

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

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

is formatted as follows.

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

To clearly present the next couple of requests, we must introduce the concept of "productive" input lines. A *productive input line* directly produces formatted output. Text lines that populate a diversion (including the top-level one) produce output, as do control lines containing requests like '`.tl //Page %//`' or an escape sequence like '`\l'1i''` (when not itself nested inside a `\w` escape sequence). Register, string, and macro definitions are not productive; neither are empty requests, nor requests and escape sequences that alter the formatting environment (as with changes to the size, face, height, slant, or color of the type). Macro calls are not themselves productive, but their interpolations can be. In GNU `troff`, if an input line by itself formats an output document, it is productive; otherwise not.[100] You can also use the `pline` request before and after an input line to see whether it produced any (additional) output nodes.[101]

---

[99] See Section 5.10 [Manipulating Hyphenation], page 114.

[100] This test does not work with AT&T `troff`, which generates device-independent output even when it does not need to.

[101] See Section 5.38 [Debugging], page 240.

We also preview the output line continuation escape sequence, \c, which "connects" two input lines that would otherwise be counted separately.[102]

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

⇒ *Chorus:* Hello, world!
⇒ Went the day well?

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

Break (unless the no-break control character is used), center the output of the next *n* productive input lines with respect to the line length and indentation, without filling, then break again regardless of the invoking control character. A nonpositive argument cancels any pending centering. Omitting the argument implies an *n* of '1'. The read-only register .ce stores the count of productive input lines remaining to be centered, which is associated with the environment (see Section 5.32 [Environments], page 224).

While the '.ad c' request also centers text, it fills the text as well.

[102] See Section 5.16 [Line Continuation], page 134.

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

The previous example illustrates a common idiom of turning centering on for a quantity of lines far in excess of what is required,[103] and off again after the text to be centered. This technique relieves humans of counting lines for requests that take a count of input lines as an argument.

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

Break (unless the no-break control character is used), align the output of the next *n* productive input lines to the right margin, without filling, then break again regardless of the control character. A nonpositive argument cancels any pending right-alignment. Omitting the argument implies an *n* of ‘`1`’. The read-only register `.rj` stores the count of productive input lines remaining to be right-aligned, which is associated with the environment (see Section 5.32 [Environments], page 224).

```
.ll 49n
.rj 3
At first I hoped that such a technically unsound
project would collapse but I soon realized it was
doomed to success. \[em] C. A. R. Hoare
    ⇒  At first I hoped that such a technically unsound
    ⇒ project would collapse but I soon realized it was
    ⇒              doomed to success. -- C. A. R. Hoare
```

[103] The `.R` register interpolates the largest value that GNU `troff` can work with. Recall Section 5.8.5 [Built-in Registers], page 105.

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

Set the sizes of spaces between words and sentences[104] in twelfths of the space width of the currently selected font.[105] (A *word space* is typically one-fourth to one-third em for Western scripts.) The default for both parameters is 12. Negative values are erroneous. The first argument is a minimum; if an output line undergoes adjustment, such spaces may increase in width. The optional second argument sets the amount of additional space separating sentences on the same output line. If omitted, this amount is set to *word-space-size*. The request is ignored if there are no parameters.

Additional inter-sentence space is used only if the output line is not full when the end of a sentence occurs in the input. If a sentence ends at the end of an input line, then both an inter-word space and an inter-sentence space are added to the output; if two spaces follow the end of a sentence in the middle of an input line, then the second space becomes an inter-sentence space in the output. Additional inter-sentence space is 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.

The read-only registers `.ss` and `.sss` hold the minimum inter-word space and supplemental inter-sentence space amounts, respectively. These parameters are part of the environment (see Section 5.32 [Environments], page 224).

The `ss` request can insert discardable horizontal space; that is, space that is discarded at a break. For example, some footnote styles collect the notes into a single paragraph with large gaps between each note.

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

If *undiscardable* space is required, use the `\h` escape sequence to put horizontal motion on the output.

---

[104] Recall Section 5.1.1 [Filling], page 67, and Section 5.1.2 [Sentences], page 68, for the definitions of word and sentence boundaries, respectively.

[105] See Section 6.1.2 [Font Description File Format], page 260. This request is incorrectly documented in the AT&T `troff` manual as using units of 1/36 em.

5.10 Manipulating Hyphenation

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

Explicitly hyphenated words such as "mother-in-law" are eligible for breaking after each of their hyphens. Relatively few words in a language offer such obvious break points, however, and automatic detection of syllabic (or phonetic) boundaries for hyphenation is not perfect,[106] particularly for unusual words found in technical literature. We can instruct GNU `troff` how to hyphenate specific words if the need arises.

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

Define each argument *word* (comprising ordinary, special, or indexed characters) as a *hyphenation exception word* such that each occurrence of a hyphen-minus '-' in *word* indicates 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 (see the `hcode` request below).

Hyphenation points specified with `hw` are not subject to the within-word placement restrictions imposed by the `hy` request (see below).

Hyphenation exception words are associated with the hyphenation language (see the `hla` request below); invoking the `hw` request in the absence of a hyphenation language is an error. Each hyphenation language maintains an independent set of hyphenation exception words.

Obtain a report of hyphenation exception words on the standard error stream with the `phw` request. See Section 5.38 [Debugging], page 240.

These are known as *hyphenation exception words* 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 word defined with `hw`.

`.rhw` [*word* . . .] [Request]

Remove each *word* from the current hyphenation language's exception word list or, without arguments, empty that list of all entries populated by the input.[107] Hyphens in *word* are ignored.

---

[106] Whether a perfect algorithm for this application is even possible is an unsolved problem in computer science: `https://tug.org/docs/liang/liang-thesis.pdf`.

[107] GNU `troff` retains those supplied by hyphenation pattern exception files; see below.

Situations also arise when only a specific occurrence of a word needs its hyphenation altered or suppressed, or when a URL or similar specialized text needs to be breakable in sensible places without hyphenation.

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

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

GNU troff regards the escape sequences \X and \Y as starting a word; that is, the \% escape sequence in, say, '\X'...'\%foobar' or '\Y'...'\%foobar' no longer prevents hyphenation of 'foobar' but inserts a hyphenation point just prior to it; most likely this isn't what you want. See Section 5.35 [Postprocessor Access], page 235.

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

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

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

.hc [*char*] [Request]

Change the hyphenation character to *char*. This character then works as the \% escape sequence normally does, and thus no longer appears in the output.[108] Without an argument, hc resets the hyphenation character to \% (the default). The hyphenation character is associated with the environment (see Section 5.32 [Environments], page 224).

.shc [*c*] [Request]

Set the *soft hyphen character*, inserted when a word is hyphenated automatically or at a hyphenation character, to the ordinary or special character *c*.[109] If the argument is omitted, the soft hyphen character is set to the default, \[hy]. If no glyph for *c* exists in the font in use at a potential hyphenation point, then the line is not broken there. Neither character definitions (specified with the char and similar requests) nor translations (specified with the tr request) are applied to *c*.

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

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

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

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

Set automatic hyphenation mode to *mode*, an integer encoding conditions for hyphenation; if omitted, the configured hyphenation mode default (see below) is implied. The hyphenation mode is available in the read-only register '`.hy`'; it is associated with the environment (see Section 5.32 [Environments], page 224). The hyphenation mode default depends on the localization file loaded when GNU `troff` starts up; see the `hpf` request below. If no localization file is loaded, the default is '`1`'.

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.[110] The entries in the following table are termed *values*; the sum of the desired values is the *mode*.

`0` disables hyphenation.

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

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.

[110] The mode is a vector of Boolean values 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.

2 disables hyphenation of the last word on a page or column,[111] even for explicitly hyphenated words.

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

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

16 enables hyphenation before the last character of a word.

32 enables hyphenation after the first character of a word.

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

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

The automatic placement of hyphens in words is determined by *pattern files*, which are derived from TEX and available for several languages. These files are named hyphen.*xx* (for the patterns) and hyphenex.*xx* (for a list of exceptions in languages that require them) where *xx* is an ISO 639 language code; see the table below.

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

[111] The formatter prevents hyphenation if the next page location trap is closer to the vertical drawing position than the next text baseline would be. See Section 5.29.1.1 [Page Location Traps], page 206. A macro package might also employ value '2' to prevent hyphenation before a display; recall Section 3.2.5 [Displays and Keeps], page 23.

A table of left and right minimum character counts for hyphenation as needed by the patterns distributed with GNU `troff` follows.[112]

language	**pattern name**	**left min**	**right min**
Czech	cs	2	2
English	en	2	3
French	fr	2	3
German traditional	det	2	2
German reformed	den	2	2
Italian	it	2	2
Polish	pl	2	2
Russian	ru	2	2
Spanish	es	2	2
Swedish	sv	1	2

Hyphenation exceptions within pattern files (that is, the words within a TEX `\hyphenation` group) obey hyphenation restrictions imposed by `hy`.

`.nh` [Request]

Disable automatic hyphenation; i.e., set the hyphenation mode to 0 (see above). The hyphenation mode of the last call to `hy` is not remembered, but invoking `hy` without an argument restores the hyphenation mode default; `groff`'s localization macro files do so for the languages listed above.

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

Set hyphenation mode default to *mode*, configuring the value the automatic hyphenation mode takes if `hy` is invoked without an argument. The hyphenation mode default is available in the read-only register '`.hydefault`'; it is associated with the environment.[113]

`.hpf` ["]*pattern-file* [Request]
`.hpfa` ["]*pattern-file* [Request]

Read hyphenation patterns from *pattern-file*, which is sought in the same way that macro files are with the `mso` request or the `-m` *`mac`* 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.

[112] See subsection "Localization packages" of *groff_tmac*(5).
[113] See Section 5.32 [Environments], page 224.

- No macro expansion is performed.
- hpf checks for the expression \patterns{...} (possibly with white-space before or after the braces). Everything between the braces is taken as hyphenation patterns. Consequently, { and } are not allowed in patterns.
- Similarly, \hyphenation{...} gives a list of hyphenation exceptions.
- \endinput is recognized also.
- For backward compatibility, if \patterns is missing, the whole file is treated as a list of hyphenation patterns (except that the % character is recognized as the start of a comment).

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

GNU troff ties the set of hyphenation patterns to the hyphenation language code selected by the hla request (see below). The hpf request is usually invoked by a localization file loaded by the troffrc file.[114]

A second call to hpf (for the same language) replaces the hyphenation patterns with the new ones. Invoking hpf or hpfa causes an error if there is no hyphenation language. If no hpf request is specified (either in the document, in a file loaded at startup, or in a macro package), GNU troff won't automatically hyphenate at all.

Caution: The hpf and hpfa requests interpret the remainder of the input line as the file name argument, including any spaces, up to a newline or comment escape sequence. Suffixing the file name with a comment, even an empty one, prevents unwanted space from creeping into it during source document maintenance.[115]

For automatic hyphenation to work, the formatter must know which letters are equivalent. For example, the letter 'E' behaves like 'e'; only the latter typically appears in hyphenation pattern files. GNU troff expects characters that participate in automatic hyphenation to be assigned *hyphenation codes* that define these equivalence classes. At startup, GNU troff assigns hyphenation codes to the letters 'a'–'z', applies the same codes to 'A'–'Z' in one-to-one correspondence, and assigns a code of zero to all other characters.

The hcode request enables application of hyphenation codes to characters outside the Unicode basic Latin set; without doing so, words containing such letters won't hyphenate properly even if the corresponding hyphenation patterns contain them. Localization files for the input character set and language configure hyphenation codes; see *groff_tmac*(5).

.hcode *dst1 src1* [*dst2 src2*] ... [Request]

Set the hyphenation code of character *dst1* to that of *src1*, and so on. *dst1* must be an ordinary character (other than a numeral) or a special

[114] For more detail on localization, see *groff_tmac*(5).

[115] See the discussion of the ds request in Section 5.22 [Strings], page 170.

character, and *src1* must be an ordinary character (other than a numeral) or a special character to which a hyphenation code has already been applied. Assigning the code of an ordinary character to itself effectively creates a unique hyphenation code (which can then be copied to others). Spaces need not separate the arguments. If any argument is invalid, `hcode` reports an error and stops reading 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 these assignments, GNU `troff` treats the German word 'Kindergärten' (the plural form of 'kindergarten') as two words 'kinderg' and 'rten' because the hyphenation code of the umlaut a is zero by default, just like a space. There is a German hyphenation pattern that covers 'kinder', so GNU `troff` finds the hyphenation 'kin-der'. The other two hyphenation points ('kin-der-gär-ten') are missed.

To remove a character's hyphenation code, copy the code of a character with a hyphenation code value of zero to it. For example, '`.hcode ß $`' removes the hyphenation code from '`ß`' (unless '`$`' has already been assigned a different one).

The `pchar` request may be helpful to troubleshoot hyphenation code assignments. See Section 5.38 [Debugging], page 240.

`.hpfcode` *a b* [*c d*] . . . [Request]

Caution: This request will be withdrawn in a future `groff` release. Use `hcode` instead.

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. `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.

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

Set the hyphenation language to *lang*, or clear it if there is no argument. Hyphenation exceptions specified with the `hw` request and hyphenation patterns and exceptions specified with the `hpf` and `hpfa` requests are associated with the hyphenation language. The `hla` request is usually invoked by a localization file, which is turn loaded by the `troffrc` or `troffrc-end` file; see the `hpf` request above.

The hyphenation language is available in the read-only string-valued register '.hla'; it is associated with the environment (see Section 5.32 [Environments], page 224).

If no hyphenation language is set or no patterns are loaded, GNU troff does not perform automatic hyphenation.

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

Set the maximum quantity of consecutive hyphenated lines to *n*. If *n* is negative, there is no maximum. If omitted, *n* is −1. This value is associated with the environment (see Section 5.32 [Environments], page 224). Only lines output from a given environment count toward the maximum associated with that environment. Hyphens resulting from \% are counted; explicit hyphens are not.

The .hlm read-only register stores this maximum. The count of immediately preceding consecutive hyphenated lines is available in the read-only register .hlc.

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

Set the (right) hyphenation margin to *length*. If the adjustment mode is not 'b' or 'n', the line is not hyphenated if it is shorter than *length*. Without an argument, the hyphenation margin is reset to its default value, 0. The default scaling unit is 'm'. The hyphenation margin is associated with the environment (see Section 5.32 [Environments], page 224).

A negative argument resets the hyphenation margin to zero.[116]

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

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

Suppress hyphenation of the line in adjustment modes 'b' or 'n' if that adjustment can be achieved 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 unit is 'm'. The hyphenation space adjustment threshold is associated with the environment (see Section 5.32 [Environments], page 224).

A negative argument resets the hyphenation space adjustment threshold to zero.[117]

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

---

[116] GNU troff also emits a warning in category 'range'. See Section 5.38.1 [Warnings], page 245.

[117] GNU troff also emits a warning in category 'range'. See Section 5.38.1 [Warnings], page 245.

5.11 Manipulating Spacing

A break causes the formatter to update the vertical drawing position at which the new text baseline is placed; you can alter this location.

`.sp` [*vertical-distance*] [Request]

Break and move the next text baseline down by *distance*, or until springing a page location trap.[118] If invoked with the no-break control character, `sp` moves the text baseline applicable to the entire pending output line by *vertical-distance*.[119] A negative *vertical-distance* cannot reduce the position of the text baseline below zero. Inside a diversion, the formatter ignores any argument. The default scaling unit is '`v`'. Omitting *vertical-distance* implies '`1v`'.

```
.pl 5v \" Set page length to 5 vees.
.de xx
\-\-\-
.  br
..
.wh 0 xx \" Set a trap at the top of the page.
foo on page \n%
.sp 2v
bar on page \n%
.sp 50v \" This will cause a page break.
baz on page \n%
.pl \n(nlu \" Truncate page to current position.
    ⇒ ---
    ⇒ foo on page 1
    ⇒
    ⇒
    ⇒ bar on page 1
    ⇒ ---
    ⇒ baz on page 2
```

The following macros place the next text baseline relative to the page top or bottom. We subtract one line height (`\n[.v]`) because the `|` operator moves the drawing position relative to the first baseline on the page (recall Section 5.4 [Numeric Expressions], page 82).

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

[118] See Section 5.29.1.1 [Page Location Traps], page 206.

[119] To shift the text baseline for *part* of an output line—to set super- or subscripts, for instance–use the `\v` escape sequence. See Section 5.25 [Page Motions], page 191.

The input '.y-from-bot-up 10c' sets the next text baseline 10 cm from the bottom edge of the paper.

Applying the boundary-relative measurement operator | operator to *vertical-distance*, as in |*N*, moves to a position relative to the page top for positive *N*, and the bottom if *N* is negative.

Invoking sp with the no-break control character while "not on a page" (in the top-level diversion at a negative vertical drawing position) is an error.[120]

.ls [*count*] [Request]
\n[.L] [Register]

Set the line spacing; add *count*−1 blank lines after each line of text. With no argument, GNU troff uses the previous value before the last ls call. The default is 1.

The read-only register .L contains the current line spacing; it is associated with the environment (see Section 5.32 [Environments], page 224).

The ls request is a coarse mechanism. See Section 5.20.1 [Changing the Type Size], page 163, for the requests vs and pvs as alternatives to ls.

```
.de SetNewLineSpacing
.  if r *old-vs .ab cannot nest SetNewLineSpacing
.  nr *old-vs \\n[.v]
.  vs (\\n[.v] * \\$1)
..
.
.de RestoreOldLineSpacing
.  vs \\n[*old-vs]
.  rr *old-vs
..
```

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

Sometimes, an output line requires additional vertical spacing, for instance to allow room for a tall construct like an inline equation with exponents or subscripts (particularly if they are iterated). The \x escape sequence takes a delimited measurement (like '\x'3p'') to increase the vertical spacing of the pending output line. The default scaling unit is 'v'. If the measurement is positive, extra vertical space is inserted below the current line; a negative measurement adds space above. If \x is applied to the pending output line multiple times, the maxima of the positive and negative adjustments are separately applied. The delimiter need not be a neutral apostrophe; see Section 5.6.5 [Delimiters], page 96.

The .a read-only register contains the extra vertical spacing *after* the text baseline of the most recently emitted output line. (In other words, it is the

[120] See Section 5.18 [Page Control], page 137, and Section 5.39.3 [Other Differences], page 251.

largest positive argument to \x encountered on that line.) This quantity is exposed via a register because if an output line requires this "extra post-vertical line spacing", and the subsequent output line requires "extra pre-vertical line spacing" (a negative argument to \x), then applying both can lead to excessive spacing between the output lines. Text that is piling high on line n might not require (as much) extra pre-vertical line spacing if line $n-1$ carries extra post-vertical line spacing.

Use of \x can be necessary in combination with the bracket-building escape sequence \b,[121] as the following example shows.

```
.nf
This is a test of \[rs]b (1).
This is a test of \[rs]b (2).
This is a test of \b'xyz'\x'-1m'\x'1m' (3).
This is a test of \[rs]b (4).
This is a test of \[rs]b (5).
    ⇒ This is a test of \b (1).
    ⇒ This is a test of \b (2).
    ⇒                   x
    ⇒ This is a test of y (3).
    ⇒                   z
    ⇒ This is a test of \b (4).
    ⇒ This is a test of \b (5).
```

Without \x, the backslashes on the lines marked '(2)' and '(4)' would be overprinted.

.ns [Request]
.rs [Request]
\n[.ns] [Register]

Enable *no-space mode*. Vertical spacing, whether by sp requests or blank input lines, is disabled. The bp request to advance to the next page is also disabled, unless it is accompanied by a page number (see Section 5.18 [Page Control], page 137). No-space mode ends automatically when text[122] is formatted for output[123] or the rs request is invoked, which ends no-space mode. The read-only register .ns interpolates a Boolean value indicating the enablement of no-space mode.

A paragraphing macro might ordinarily insert vertical space to separate paragraphs. A section heading macro could invoke ns to suppress this spacing for the first paragraph in a section.

---

[121] See Section 5.27 [Drawing Geometric Objects], page 200.
[122] or geometric objects; see Section 5.27 [Drawing Geometric Objects], page 200
[123] to the top-level diversion; see Section 5.30 [Diversions], page 217

5.12 Tabs and Fields

A tab character (code point 9) causes a horizontal movement to the next tab stop, if any.

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

Set tab stop positions. This request takes a series of tab specifiers as arguments (optionally divided into two groups with the letter '`T`') that indicate where each tab stop is to be, overriding any previous settings. The default scaling unit is '`m`'. Invoking `ta` without arguments removes all tab stops. GNU `troff`'s startup value is '`T 0.5i`'.

Tab stops can be specified absolutely—as distances from the left margin. The following example sets six tab stops, one every inch.

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

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

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

GNU `troff` supports an extended syntax to specify repeating tab stops. These stops appear after a '`T`' argument. Their values are always taken as distances relative to the previous tab stop. This is the idiomatic way to specify tab stops at equal intervals in `groff`. The following is, yet again, the same as the previous examples. It does more, in fact, since it defines an infinite number of tab stops at one-inch intervals.

```
.ta T 1i
```

Now we are ready to interpret the full syntax given above. The `ta` request sets tabs at positions *n1*, *n2*, . . . , *nn*, then at *nn*+*r1*, *nn*+*r2*, . . . , *nn*+*rn*, then at *nn*+*rn*+*r1*, *nn*+*rn*+*r2*, . . . , *nn*+*rn*+*rn*, and so on. For example, '`4c +6c T 3c 5c 2c`' is equivalent to '`4c 10c 13c 18c 20c 23c 28c 30c ...`'.

Text between two tab stops may be aligned to the right or left, or centered. This alignment is determined by appending '`R`', '`L`', or '`C`' to the tab specifier. The default is '`L`'.

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

The beginning of an output line is not a tab stop; the text that begins an output line is placed according to the configured alignment and indentation; see Section 5.9 [Manipulating Filling and Adjustment], page 106, and Section 5.15 [Line Layout], page 131.

A tab stop becomes a non-breakable horizontal movement that cannot be adjusted.

```
.ll 2i
.ta T 1i
a→b→c
    error  warning: cannot adjust line; overset by 1n
    ⇒ a         b         c
```

The above creates a single output line that is a bit longer than two inches. Now consider the following.

```
.ll 2i
.ta T 1i
a→b c→d
    error  warning: cannot adjust line; underset by 9n
    ⇒ a         b
    ⇒ c       d
```

GNU `troff` first converts the line's tab stops into unbreakable horizontal movements, then breaks after '`b`'. This usually isn't what you want.

Superfluous tab characters—those that do not correspond to a tab stop—are ignored except for the first, which delimits the characters belonging to the last tab stop for right-alignment or centering.

```
.nf
.ta 2i 4iR
\l'4i\&-'
foo→bar
foo→bar→baz
foo→bar→bazqux
foo→bar→baz→qux
    ⇒ ----------------------------------------
    ⇒ foo                 bar
    ⇒ foo                 bar              baz
    ⇒ foo                 bar           bazqux
    ⇒ foo                 bar              bazqux
```

We see that "bar" is between the first and second tab stops, not the second and (nonexistent) third. The first "baz" is right-aligned within the second tab stop. The second is catenated with "qux" and right-aligned within it. The third "baz" is aligned like the first because the tab character after it determines the right boundary of the tab stop.

Tab stops are associated with the environment (see Section 5.32 [Environments], page 224).

The read-only register `.tabs` contains a string representation of the current tab settings suitable for use as an argument to the `ta` request.[124]

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

`\t` [Escape sequence]

Interpolate a tab in copy mode.[125]

[124] DWB 3.3, Plan 9, and Heirloom Doctools `troff`s uses the register `.S` for this purpose, lack, and thus do not report, GNU `troff`'s `T` extension to tab stop syntax.

[125] See Section 5.24.2 [Copy Mode], page 188.

`.tc` [*c*] [Request]

Set the tab repetition character to the ordinary or special character *c*; normally, no glyph is written when moving to a tab stop (and some output devices may output space characters to achieve this motion). A *tab repetition character* causes the formatter to write as many instances of *c* as are necessary to occupy the interval from the horizontal drawing position to the next tab stop. If *c* is omitted, GNU `troff` reverts to the default behavior. The tab repetition character is associated with the environment.[126] Only a single character of *c* is recognized; any excess is ignored.

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

Activate or deactivate line-tabs in the environment per Boolean expression *b*. They are inactive by default, and activated if *b* is omitted. When line-tabs are active, tab stops are computed relative to the start of the pending output line instead of the drawing position corresponding to the start of the input line.

```
.ta 1i 3i
a→\c
b→\c
c
.br
.linetabs
a→\c
b→\c
c
    ⇒ a         b         c
    ⇒ a         b                   c
```

The read-only register `.linetabs` interpolates 1 if line-tabs are active, and 0 otherwise.

5.12.1 Leaders

Sometimes it is desirable to fill a tab stop with a given glyph, but also use tab stops normally on the same output line. An example is a table of contents entry that uses dots to bridge the entry name with its page number, which is itself aligned between tab stops. The `roff` language provides *leaders* for this purpose.[127]

A leader character (code point 1, also known as SOH or "start of heading"), behaves similarly to a tab character: it moves to the next tab stop. The difference is that for this movement, the default fill character is a period '`.`'.

[126] See Section 5.32 [Environments], page 224.

[127] Pronounce "leader" to rhyme with "feeder"; it refers to how the glyphs "lead" the eye across the page to the corresponding page number or other datum.

`\a` [Escape sequence]
Interpolate a leader in copy mode; see Section 5.24.2 [Copy Mode], page 188.

`.lc` [*c*] [Request]
Set the leader repetition character to the ordinary or special character *c*. Recall Section 5.1.6 [Tabs and Leaders], page 71: when encountering a leader character in the input, the formatter writes as many dots '`.`' as are necessary until reaching the next tab stop; this is the *leader definition character*. Omitting *c* unsets the leader character. With no argument, GNU `troff` treats leaders the same as tabs. The leader repetition character is associated with the environment (see Section 5.32 [Environments], page 224). Only a single *c* is recognized; any excess is ignored.

A table of contents, for example, may define tab stops after a section number, a title, and a gap to be filled with leader dots. The page number follows the leader, after a right-aligned final tab stop wide enough to house the largest page number occurring in the document.

```
.ds entry1 19.\tThe Prophet\a\t98
.ds entry2 20.\tAll Astir\a\t101
.ta .5i 4.5i +.5iR
.nf
\*[entry1]
\*[entry2]
    ⇒ 19.  The Prophet.............................   98
    ⇒ 20.  All Astir...............................  101
```

5.12.2 Fields

Fields are a more general way of laying out tabular data. A field is defined as the data between a pair of *delimiting characters*. It contains substrings that are separated by *padding characters*. The width of a field is the distance on the *input* line from the position where the field starts to the next tab stop. A padding character inserts an adjustable space similar to TEX's `\hss` command (thus it can even be negative) to make the sum of all substring lengths plus the adjustable space equal to the field width. If more than one padding character is inserted, the available space is evenly distributed among them.

`.fc` [*delim-char* [*padding-char*]] [Request]
Define a delimiting and a padding character for fields. If the latter is missing, the padding character defaults to a space character. If there is no argument at all, the field mechanism is disabled (which is the default). In contrast to, e.g., the tab repetition character, delimiting and padding characters are *not* associated with the environment (see Section 5.32 [Environments], page 224).

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

5.13 Character Translations

A *character translation* remaps input characters, usually to make input more convenient. GNU troff performs the translation just before formatting the glyph; that is, right before it converts the character's token to a node.[128]

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

Translate character *a* to *b*, *c* to *d*, and so on prior to output. Unusually, spaces are not permitted between the arguments. If number of arguments is odd, the formatter translates the last to a fixed-width space (the same one obtained by the \SPC escape sequence).

The trin request works as does tr, except that asciify (see Section 5.30 [Diversions], page 217) ignores the translation when a diversion is interpolated.

Some notes:

- Special characters (\(*xx*, \[*xxx*], \C'*xxx*', \', \`, \-, _), glyphs defined with the char request, and numbered glyphs (\N'*xxx*') can be translated also.
- The \e escape can be translated also.
- Characters can be mapped onto the \% and \~ escape sequences (but \% and \~ can't be mapped onto another glyph).
- The following characters can't be translated: space (with one exception, see below), backspace, newline, leader (and \a), tab (and \t).
- Translations are not considered for finding the soft hyphen character set with the shc request.
- The pair '*c*\&' (an arbitrary character *c* followed by the dummy character) maps this character to "nothing".

  ```
  .tr a\&
  foo bar
      ⇒ foo br
  ```

  Even the space character can be mapped to the dummy character.

[128] See Section 5.37 [GNU troff Internals], page 237, for more on this process.

```
.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 adjustment is enabled, it occurs in spite of the 'empty' space character; but no minimum distance—no minimum inter-word space—separates 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 elements of its argument exist.

  See Section 5.37 [GNU troff Internals], page 237.
- Without an argument, the `tr` request is ignored.

`.trnt` *abcd...* [Request]

`trnt` is the same as the `tr` request except that the translations do not apply to text that is transparently throughput into a diversion with `\!`. See Section 5.30 [Diversions], page 217.

For example,

```
.tr ab
.di x
\!.tm a
.di
.x
```

prints '`b`' to the standard error stream; if `trnt` is used instead of `tr` it prints '`a`'.

5.14 troff and nroff Modes

Historically, `nroff` and `troff` were two separate programs; the former for terminal output, the latter for typesetters. GNU `troff` merges both functions into one executable[129] that sends its output to a device driver (`grotty` for terminal devices, `grops` for PostScript, and so on) that interprets its output. 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

[129] A GNU `nroff` program is available for convenience; it runs GNU `troff` to perform formatting; see *nroff*(1).

specially if a terminal output device is used, so such a strong distinction is unnecessary.

Usually, a macro package can be used with any output device. Nevertheless, it is sometimes necessary to distinguish terminal and non-terminal devices: the formatter provides the conditional expressions[130] 'n' and 't' to help documents format as desired for these device classes, which have divergent capabilities. AT&T nroff silently ignores requests and escape sequences that change the type size, for example.

GNU troff furthermore offers requests that configure it to behave like nroff or like troff. This feature spares the document author the tedium of guarding changes of type size or font family with 'if t' control lines or similar.

.troff [Request]

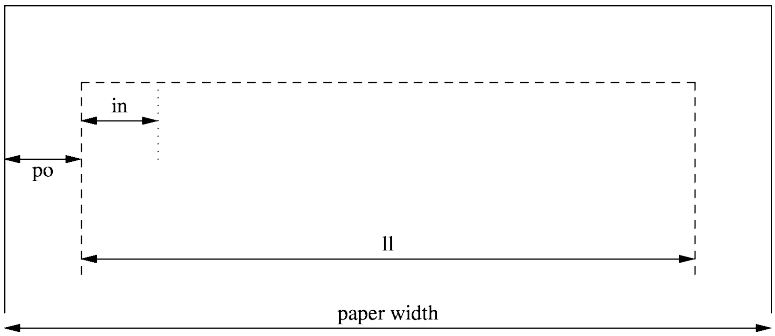
Make the 't' built-in condition true (and the 'n' built-in condition false) in conditional expressions. This is the default. However, GNU troff normally reads the startup files troffrc and troffrc-end, which can alter it. Recall -R from Section 2.1 [Groff Options], page 7.

.nroff [Request]

Make the 'n' built-in condition true (and the 't' built-in condition false) in conditional expressions. The file tty.tmac switches GNU troff to nroff mode; troffrc loads it if the name of the output device matches that of a known "typewriter-like device"; recall Section 2.1 [Groff Options], page 7.

5.15 Line Layout

The following diagram shows the dimensions that the formatter uses to arrange a line of output on the page. Each dimension is labelled with the name of the request that configures it.



The dimensions are defined as follows.

[130] See Section 5.23 [Conditionals and Loops], page 175.

`po` The *page offset* is the leftmost position of running text.

`in` *Indentation* is the distance from the page offset at which text is set.

`ll` *Line length* is the maximum extent of unindented running text.

The page offset can be thought of as the "left margin". The right margin is not explicitly configured; the combination of page offset and line length provides the information necessary to derive it.

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

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

Requests exist to place line numbers and margin characters within the page margins.[131]

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

The `po` request sets the page offset to *offset*; if *offset* is signed, it adjusts the page offset by its value. The default scaling unit is '`m`'. The default offset is 1 i on typesetters and zero on terminals. If *offset* is omitted, the formatter resets the page offset to that before the previous invocation of `po`, and zero if there is none. Read-only register '`.o`' interpolates the page offset.

This request is incorrectly documented in the AT&T `troff` manual as using a default scaling unit of '`v`'.

[131] See Section 5.26 [Output Line Annotation], page 197.

```
.po 3i
\n[.o]
    ⇒ 720
.po -1i
\n[.o]
    ⇒ 480
.po
\n[.o]
    ⇒ 720
```

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

Set indentation to *indent*; if *indent* is signed, adjust the indentation by its value. The default scaling unit is '`m`'. Initially, there is no indentation. This request causes a break. The indentation amount is associated with the environment.[132]

If *indent* is omitted, the formatter resets the indentation to that before the previous invocation of `in`, and zero if there is none. If *indent* is negative, the formatter sets the indentation (and temporary indentation; see below) to zero.[133]

The formatter delays the effect of `in` until it has emitted any partially collected line. In other words, `in` does not change a pending output line's indentation.

The read-only register '`.i`' interpolates the indentation amount, ignoring temporary indentation (see below).

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

Temporarily indent the next output line by *amount*; if *amount* is signed, apply it as an offset to the indentation configured by the `in` request. The default scaling unit is '`m`'. This request causes a break. The effect of `ti` is delayed until a partially collected line (if it exists) is output. In other words, it does not change a pending output line's indentation. The read-only register `.in` reports the total indentation that applies to the pending output line. The temporary indentation is associated with the environment.[134]

If *amount* is omitted, GNU `troff` cancels any pending temporary indentation.

---

[132] See Section 5.32 [Environments], page 224.

[133] GNU `troff` emits a warning in category '`range`'. See Section 5.38.1 [Warnings], page 245.

[134] See Section 5.32 [Environments], page 224.

Idiomatic applications of the `ti` request are invocations by paragraphing macros; (1) to set up a first-line indentation of positive sign; and (2) to set up a "normal" temporary indentation of negative sign offsetting a positive indentation that applies to the whole parargraph. The latter combination is known as a "hanging indent".

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

Change (increase or decrease) the line length per the numeric expression *length*. The default scaling unit is '`m`'. If not otherwise configured (recall Section 2.5 [Paper Format], page 16), the default line length is 6.5 i. If *length* is invalid, the formatter ignores the request. If *length* is nonpositive, the formatter sets the line length to the device's horizontal motion quantum; recall Section 5.3.1 [Motion Quanta], page 81.[135] The line length is associated with the environment.[136] If *length* is omitted, the formatter restores the environment's previous line length.

The effect of `ll` is delayed until a partially collected line (if it exists) is output. In other words, it does not change a pending output line's length.

The line length as set by `ll` can be found in the read-only register '`.l`'. The read-only register `.ll` reports the line length that applies to the pending output line. Similarly 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 previous length.

5.16 Line Continuation

When filling is enabled, input and output line breaks generally do not correspond. The `roff` language therefore distinguishes input and output line continuation.

`\RET` [Escape sequence]

`\RET` (a backslash immediately followed by a newline) suppresses the effects of that newline in the input. The next input line thus retains the classification of its predecessor as a control or text line. `\RET` is useful for managing line lengths in the input during document maintenance; you can even break an input line in the middle of a word, request invocation, macro call, or escape sequence. Input line continuation is invisible to the formatter, with two exceptions: the `|` operator recognizes the new input line (see Section 5.4 [Numeric Expressions], page 82), and the input line

[135] GNU `troff` emits a warning in category '`range`'. See Section 5.38.1 [Warnings], page 245.

[136] See Section 5.32 [Environments], page 224.

counter register .c increments. The formatter interprets \RET even in copy mode.[137]

```
.ll 50n
.de I
.  ft I
.  nop \\$*
.  ft
..
Our film class watched
.I The Effect of Gamma Rays on Man-in-the-Moon
Marigolds. \" whoops, the input line wrapped
.br
.I My own opus begins on line \n[.c] \
and ends on line \n[.c].
```

⇒ Our film class watched *The Effect of Gamma Rays on*
⇒ *Man-in-the-Moon* Marigolds.
⇒ *My own opus begins on line 11 and ends on line 12.*

\c [Escape sequence]
\n[.int] [Register]

\c continues an output line. Nothing after it on the input line is formatted. In contrast to \RET, a line after \c remains a new input line, so a control character is recognized at its beginning. The visual results depend on whether filling is enabled; see Section 5.9 [Manipulating Filling and Adjustment], page 106.

- If filling is enabled, a word interrupted with \c is continued with the text on the next input text line, without an intervening space.

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

⇒ This is a test.

- If filling is disabled, the next input text line after \c is handled as a continuation of the same input text line.

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

⇒ This is a test.

An intervening control line that causes a break overrides \c, flushing out the pending output line in the usual way.

The .int read-only register interpolates a positive value only if the pending output line has been continued with \c; this datum is associated with the environment.[138]

---

[137] See Section 5.24.2 [Copy Mode], page 188.

[138] See Section 5.32 [Environments], page 224. Historically, the \c escape sequence has proven challenging to characterize. Some sources say it "connects the next input text"

5.17 Page Layout

The formatter permits configuration of the page length and page number.

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

Change (increase or decrease) the page length per the numeric expression *length*. The default scaling unit is '`v`'. If *length* is invalid, GNU `troff` emits a warning in category '`number`'. If *length* is absent or invalid, '`11i`' is assumed. If *length* is nonpositive, GNU `troff` emits a warning in category '`range`' and sets the page length to the device's vertical motion quantum; recall Section 5.3.1 [Motion Quanta], page 81.

The read-only register '`.p`' interpolates the current page length.

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

Change (increase or decrease) the page number of the *next* page per the numeric expression *num*. If *num* is invalid, GNU `troff` emits a warning in category '`number`' and ignores the request. Without an argument, `pn` is ignored.

The read-only register `.pn` interpolates *num* if set by `pn` on the current page, or the current page number plus 1.

The formatter offers special support for typesetting headers and footers, collectively termed *titles*. Titles have an independent line length, and their placement on the page is not restricted.

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

Format an output line as a title consisting of *left*, *center*, and *right*, each aligned accordingly. The delimiter need not be a neutral apostrophe: `tl` accepts the same delimiters as most escape sequences; see Section 5.6.5 [Delimiters], page 96. If not used as the delimiter, any *page number character* character is replaced with the current page number; the default is '`%`'; see the the `pc` request below. Without an argument, `tl` is ignored. `tl` writes the title line immediately, ignoring any partially collected line.

It is not an error to omit delimiters after the first. For example, '`.tl /Thesis`' is interpreted as '`.tl /Thesis///`': it sets a title line comprising only the left-aligned word '`Thesis`'.

(to the input line on which it appears); others describe it as "interrupting" text, on the grounds that a text line is interrupted without breaking, perhaps to inject a request invocation or macro call.

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

Change (increase or decrease) the line length used by titles per the numeric expression *length*. The default scaling unit is 'm'. The formatter's default title length is '6.5i'. If *length* is invalid, GNU troff emits a warning in category 'number' and ignores the request. If *length* is non-positive, GNU troff emits a warning in category 'range' and sets the title line length to the device's horizontal motion quantum; recall Section 5.3.1 [Motion Quanta], page 81. The title length is is associated with the environment (see Section 5.32 [Environments], page 224). If *length* is omitted, GNU troff restores the environment's previous title length.

The read-only register '.lt' interpolates the title line length.

.pc [*char*] [Request]

Set the page number character to *char*. With no argument, the page number character is disabled. pc does not affect the register %.

The following example exercises title features.

```
.lt 50n
This is my partially collected
.tl 'Isomers 2023'%'Dextrose Edition'
line.
    ⇒ Isomers 2023            1        Dextrose Edition
    ⇒ This is my partially collected line.
```

We most often see titles used in page header and footer traps. See Section 5.29 [Traps], page 206.

5.18 Page Control

Discretionary page breaks can prevent the unwanted separation of content. A new page number takes effect during page ejection; see Section 5.29.1.2 [The Implicit Page Trap], page 211.

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

Break the page and change (increase or decrease) the next page number per the numeric expression *page-number*. If *page-number* is invalid, GNU troff emits a warning in category 'number' and ignores the argument. This request causes a break. A page break advances the vertical drawing position to the bottom of the page, springing traps. See Section 5.29.1.1 [Page Location Traps], page 206. bp has effect only if invoked within

the top-level diversion.[139] This request is incorrectly documented in the AT&T `troff` manual as having a default scaling unit of '`v`'.

The register `%` interpolates the page number.

```
.de BP
'  bp \" schedule page break once current line is output
..
```

Caution: Interpolations occur before formatting operations. The process of filling, breaking, and adjusting a line can change the page number. `%` is a register like any other, not a placeholder that is rewritten after the line it appears on is formatted. Consider, for example, an extremely long page number at the end of the last line on the page; numbers aren't hyphenated, so the word containing the page number might break the line and the page, causing the reported page number to lag by one. This sequencing also means that interpolating the `%` register inside a diversion (such as a footnote) records the page number at the time the diversion is populated, not when it is output.

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

Force a page break if insufficient vertical space is available; it asserts "needed" space. `ne` tests the distance to the next page location trap; see Section 5.29.1.1 [Page Location Traps], page 206, and breaks the page if that amount is less than *space*. The default scaling unit is '`v`'. If *space* is invalid, GNU `troff` emits a warning in category '`number`' and ignores the argument. If *space* is not specified, '`1v`' is assumed.

We can require space for at least the first two output lines of a paragraph, preventing its first line from being isolated at the page bottom.

```
.ne 2v
Considering how common illness is,
how tremendous the spiritual change that it brings,
how astonishing,
when the lights of health go down,
the undiscovered countries that are then disclosed,
what wastes and deserts of the soul a slight attack
of influenza brings to view,
what precipices and lawns sprinkled with bright flowers
a little rise of temperature reveals,
what ancient and obdurate oaks are uprooted in us
in the act of sickness,
how we go down into the pit of death
and feel the waters of annihilation
close above our heads.\|.\|.
.sp
Virgina Woolf, \[lq]On Being Ill\[rq], 1926
```

[139] See Section 5.30 [Diversions], page 217.

This method is reliable only if no output line is pending when ne is invoked. When macro packages are used, this is often not the case: their paragraphing macros perform the break. You may need to experiment with placing the ne after the paragraphing macro, or br and ne before it.

ne is also useful to force grouping of section headings with their subsequent paragraphs, or tables with their captions and/or explanations. Macro packages often use ne with diversions to implement keeps and displays; see Section 5.30 [Diversions], page 217. They may also offer parameters for widow and orphan management.

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

sv requires vertical space as ne does, but also *saves* it for later output by the os request. If *space* is available before the next page location trap, it is output immediately. Both requests ignore a partially collected line, taking effect at the next break. sv and os ignore no-space mode (recall Section 5.11 [Manipulating Spacing], page 122). While the sv request allows negative values for *space*, os ignores them. The default scaling unit is 'v'. If *space* is not specified, '1v' is assumed.

\n[nl] [Register]

nl interpolates the vertical drawing position as of the most recently typeset output line.[140] It does not necessarily (and often does not) represent that of the pending output line, because the formatter does not determine the position of its baseline until it is output; recall Section 5.11 [Manipulating Spacing], page 122. Assigning a value to nl sets the vertical drawing position in advance of further modifications to baseline positioning arising from alterations to type size, changes to vertical spacing, or application of extra pre- or post-vertical spacing.

When the formatter starts, the transition to the first page has not yet happened—nl is negative. If you plant a page location trap at vertical position '0' (idiomatically to format a header), you can assign a negative value to nl to spring that trap even if the page has already started.[141]

[140] See Section 5.30 [Diversions], page 217, for a comparison of nl with the .h and .d registers.

[141] See Section 5.29.1.1 [Page Location Traps], page 206.

```
.de HD
.  sp
.  tl ''Goldbach Solution''
.  sp
..
.
First page.
.bp
.wh 0 HD \" plant header trap at top of page
.nr nl (-1)
Second page.
```

```
    ⇒ First page.
    ⇒
    ⇒ (blank lines elided)
    ⇒
    ⇒                          Goldbach Solution
    ⇒
    ⇒ (blank lines elided)
    ⇒
    ⇒ Second page.
```

Without resetting `nl` to a negative value, the trap just planted would be active beginning with the *next* page, not the current one.

5.19 Using Fonts

In digital typography, a *font* is a collection of characters in a specific typeface that a device can render as glyphs at a desired size.[142] A `roff` formatter can change typefaces at any point in the text. The basic faces are a set of *styles* combining *upright* and *slanted* (italic or oblique) shapes with normal and heavy stroke weights: '`R`', '`I`', '`B`', and '`BI`'—these stand for *roman*, *italic*, *bold*, and *bold-italic*. For linguistic text, GNU `troff` groups typefaces into *families* containing each of these styles.[143] A *text font* is thus often a family combined with a style, but it need not be: consider the `ps` and `pdf` devices' `ZCMI` (Zapf Chancery Medium italic)—often, no other style of Zapf Chancery Medium is provided. On typesetters, at least one *special font* is available, comprising *unstyled* glyphs for mathematical operators and other purposes.

Like the AT&T `troff` formatter, GNU `troff` does not itself load or manipulate a digital font file;[144] instead it works with a *font description file* that characterizes it, including its glyph repertoire and the *metrics* (dimensions)

[142] Terminals and some typesetters have fonts that render at only one or two sizes. As examples, take the `groff lj4` device's Lineprinter, and `lbp`'s Courier and Elite faces.

[143] Font designers prepare families such that the styles share esthetic properties.

[144] Historically, the fonts `troff`s dealt with were not Free Software or, as with the Graphic Systems C/A/T, did not even exist in the digital domain.

of each glyph.[145] This information permits the formatter to accurately place glyphs with respect to each other. Before using a font description, the formatter associates it with a *mounting position*, a place in an ordered list of available typefaces. So that a document need not be strongly coupled to a specific font family, in GNU troff an output device can associate a style in the abstract sense with a mounting position. Thus the default family can be combined with a style dynamically, producing a *resolved font name*. A user-specified font name that combines family and style, or refers to a font that is not a member of a family, is already "resolved".

Fonts often have trademarked names, and even Free Software fonts can require renaming upon modification. groff maintains a convention that a device's serif font family bears the name 'T' ("Times"), its sans-serif family 'H' ("Helvetica"), and its monospaced family 'C' ("Courier"). Historical inertia has driven groff's font identifiers to short uppercase abbreviations of font names, as with 'TR', 'TI', 'TB', 'TBI', and a special font 'S'.

The default family used with abstract styles can be changed at any time; initially, it is 'T'. Typically, device description files arrange abstract styles in the first four mounting positions in the order shown above.[146] The default mounting position, and therefore style, is always '1' ('R'). By issuing appropriate formatter instructions, you can override these defaults before your document writes its first glyph.

Terminals cannot change font families and lack special fonts. They support style changes by overstriking, or by altering ISO 6429/ECMA-48 *graphic renditions* (character cell attributes).

5.19.1 Selecting Fonts

We use *font* to refer to any of several means of identifying a typeface: by its mounting position ('3'), by its identifier ('TB'), or by an abstract style ('B') to be combined with the default family.

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

Select the typeface *font*. If *font* is a non-negative integer, the formatter interprets it as a mounting position; the font mounted there is selected. If that position refers to an abstract style, GNU troff combines it with the default family (see fam and \F below) to make a resolved font name. If *font* is 'DESC', if the mounting position is not an abstract style and no font is mounted there, or if the mounting position is negative, the formatter

[145] See Section 6.1.2 [Font Description File Format], page 260.
[146] See Section 6.1.1 [DESC File Format], page 257.

ignores the request.[147] If the argument is absent or 'P', the formatter selects the previously used typeface and makes the current selection the previous one; if there is no previous font selection, the formatter ignores the request. These data are associated with the environment.[148]

The resolved font name is subject to translation (see request `ftr` below). Next, the (possibly translated) font name's mounting position is looked up; if not mounted, *font* is sought on the file system as a font description file and, if located, automatically mounted at the next available position (see register `.fp` below). If the font was mounted using an identifier different from its font description file name (see request `fp` below), that file name is then sought. If a font description file for the resolved font name is not found, GNU `troff` emits a warning in category '`font`' and ignores the request.

The `\f` escape sequence is similar, accepting names or mounting positions of one character *f*, two characters *fn*, or arbitrary length *font*. '`\f[]`' selects the previous font. GNU `troff` supports the syntax form '`\fP`' for compatibility with AT&T `troff`, and '`\f[P]`' for consistency.

```
eggs, bacon,
.ft I
spam,
.ft
and sausage.
.br
eggs, bacon, \fIspam,\fP and sausage.
    ⇒ eggs, bacon, spam, and sausage.
    ⇒ eggs, bacon, spam, and sausage.
```

The read-only string-valued register `.fn` contains the resolved font name of the selected font. Copy its value to a string to save it for later use.

```
.ds saved-font \n[.fn]
. . . text involving many font changes . . .
.ft \*[saved-font]
```

The formatter does not tokenize `\f` when reading it; it thus can be used in requests that expect a single-character argument. We can assign a font to a margin character as follows.[149]

```
.mc \f[I]x\f[]
```

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

Translate font name *f* to *g*. Where the `\f` escape sequence, the `F` and `S` conditional expression operators, and the `ft`, `ul`, `bd`, `cs`, `tkf`, `special`, `fspecial`, `fp`, or `sty` requests refer to *f*, GNU `troff` uses *g* instead. Omit *g* or repeat *f* as *g* to untranslate *f*. *f* and *g* need not be mounted fonts.

[147] GNU `troff` emits a warning in category '`font`' or '`range`', as appropriate. See Section 5.38.1 [Warnings], page 245.

[148] See Section 5.32 [Environments], page 224.

[149] See Section 5.26 [Output Line Annotation], page 197.

Obtain a report of font translations defined by `ftr` on the standard error stream with the `pftr` request.[150]

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

Set magnification of *font* to factor *zoom*, a multiplier in thousandths applied to the type size. *zoom* must be non-negative. `fzoom` applies to glyphs when they are formatted, altering a font's apparent size in relation to others. If *zoom* is missing or equal to zero or 1000, *font* is not magnified. *font* must be a resolved font name, not an abstract style or a mounting position.

Font magnification is transparent to some aspects of GNU `troff`. A change of the zoom factor affects scaling of glyph sizes, inter-word and inter-sentence spaces, and kerning adjustments on the output device, but *not* vertical spacing. It is not reflected in registers that report the requested or current type size, or the minimum inter-word and supplemental inter-sentence space sizes. It *is* reflected in measurements of formatted output: the horizontal drawing position register `hp`, interpolation of the `\w` escape sequence, and the registers updated by that escape sequence or the formatting of a glyph in the environment.[151]

`fzoom` can harmonize the apparent cap-heights of fonts from different families when formatted on the same baseline at the same type size.

```
.fzoom HR 900
.fzoom CR 1150
.fzoom PR 950
Times, \F[H]Helvetica\F[], \F[C]Courier\F[],
and \F[P]Palatino\F[].
.sp
M\F[H]M\F[C]M\F[P]M
```

The zoom factor of the currently selected font is available in the read-only register '`.fzoom`'. It interpolates zero if there is no magnification.

5.19.2 Font Families

To accommodate the wide variety of fonts available, GNU `troff` distinguishes *font families* and *font styles*. A resolved font name is the catenation of a font family and a style. Selecting an abstract style causes GNU `troff` to combine it with the default font family.

You can thus compose a document using abstract styles exclusively for its body or running text—selecting a specific family only for titles or examples, for instance—and change the default family on the command line.

[150] See Section 5.38 [Debugging], page 240.
[151] See Section 5.32 [Environments], page 224.

`.fam` [*family*] [Request]
`\n[.fam]` [Register]
`\F`*f* [Escape sequence]
`\F(`*fm* [Escape sequence]
`\F[`*family*`]` [Escape sequence]

Set the default font family, used in combination with abstract styles to construct a resolved font name, to *family* (one-character name *f*, two-character name *fm*). Without an argument, GNU `troff` selects the previous font family; if there is none, it falls back to the device's default[152] or its own ('`T`').

The `\F` escape sequence works similarly. In disanalogy to `\f`, '`\FP`' makes '`P`' the default family. Use '`\F[]`' to select the previous default family. The default font family is available in the read-only string-valued register `.fam`; this datum is associated with the environment.[153]

```
spam,      \" startup defaults are T (Times) R (roman)
.fam H     \" make Helvetica the default family
spam,      \" family H + style R = HR
.ft B      \" family H + style B = HB
spam,
.ft CR     \" Courier roman (default family not changed)
spam,
.ft        \" back to Helvetica bold
spam,
.fam T     \" make Times the default family
spam,      \" family T + style B = TB
.ft AR     \" font AR (not a style)
baked beans,
.ft R      \" family T + style R = TR
and spam.
```

GNU `troff` does not tokenize `\F` when reading it; it thus can be used in requests that expect a single-character argument. We can assign a font family to a margin character as follows.[154]

```
.mc \F[P]x\F[]
```

`.sty` *pos style* [Request]
`\n[.sty]` [Register]

Associate an abstract style *style* with mounting position *pos*, which must be a non-negative integer. Applying the requests `cs`, `bd`, `tkf`, `uf`, or `fspecial` to an abstract style affects the member of the default family corresponding to that style.

GNU `troff`'s `-f` command-line option (recall Section 2.1 [Groff Options], page 7) and `fam` request set the default family. The `styles` directive in

[152] See Section 6.1.1 [DESC File Format], page 257.
[153] See Section 5.32 [Environments], page 224.
[154] See Section 5.26 [Output Line Annotation], page 197.

the output device's DESC file controls which mounting positions (if any) initially associate with abstract styles rather than fonts, and the sty request updates this association.

Caution: GNU troff does not validate the *style* argument. Errors may occur later, when the formatter attempts to construct a resolved font name, or format a character for output.

```
.nr BarPos \n[.fp]
.sty \n[.fp] Bar
.fam Foo
.ft \n[BarPos]
.tm .f=\n[.f]
A
```

```
error  error: no font family named 'Foo' exists
error  .f=41
error  error: cannot format glyph: no current font
```

When an abstract style has been selected, the read-only string-valued register '.sty' interpolates its name; this datum is associated with the environment.[155] Otherwise, '.sty' interpolates nothing.

5.19.3 Font Mounting Positions

To support typeface indirection through abstract styles, and for compatibility with AT&T troff, the formatter maintains a list of *positions* at which fonts required by a document are *mounted*. An output device's description file DESC typically configures a set of pre-mounted fonts.[156] A font need not be explicitly mounted before it is selected; GNU troff searches GROFF_FONT_PATH for a file name matching the identifier and mount it on demand.

.fp *pos id* [*font-description-file-name*] [Request]
\n[.f] [Register]
\n[.fp] [Register]

Mount a font under the name *id* at mounting position *pos*, a non-negative integer. When the formatter starts up, it reads the output device's description to mount an initial set of faces, and selects font position 1. Position 0 is unused by default. Unless the *font-description-file-name* argument is given, *id* should be the name of a font description file stored in a directory corresponding to the selected output device. GNU troff does not traverse directories to locate the font description file.

The optional third argument enables font names to be aliased, which can be necessary in compatibility mode[157] since AT&T troff syntax affords no means of identifying fonts with names longer than two characters, like 'TBI' or 'ZCMI', in a font selection escape sequence. You can also alias

[155] See Section 5.32 [Environments], page 224.
[156] See Section 6.1 [Device and Font Description Files], page 257.
[157] See Section 5.39.2 [Compatibility Mode], page 248.

fonts on mounting for convenience or abstraction. (See below regarding the `.fp` register.)

```
.fp \n[.fp] SC ZCMI
Send a \f(SChand-written\fP thank-you note.
.fp \n[.fp] Emph TI
.fp \n[.fp] Strong TB
Are \f[Emph]these names\f[] \f[Strong]comfortable\f[]?
```

'`DESC`', '`P`', and non-negative integers are not usable as font identifiers.

Obtain a report of occupied font mounting positions (whether configured by the '`DESC`' file, the `fp` request, or automatic mounting) on the standard error stream with the `pfp` request.[158]

The position of the currently selected font (or abstract style) is available in the read-only register '`.f`'.

Copy the value of `.f` to another register to save it for later use.

```
.nr sF \n(.f
... text involving many font changes ...
.ft \n(sF
```

The index of the next (non-zero) free font mounting position is available in the read-only register '`.fp`'. When using the `ft` request or `\f` escape sequence to select a font by name that is not listed in the device's `DESC` file, GNU `troff` automatically mounts it at position '`\n[.fp]`'. If you mount a font at a position explicitly with the `fp` request, you should follow the same practice, though the program does not strictly enforce adherence.[159]

5.19.4 Characters and Glyphs

A glyph is a graphical representation of a character. Whereas a *character* is an abstraction of semantic information, a *glyph* is an intelligible mark visible on screen or paper. A character has many possible representation forms; for example, the character 'A' can be written in an upright or slanted typeface, producing distinct glyphs. Sometimes, a sequence of characters map to a single glyph: this is a *ligature*—the most common is 'fi'.

Space characters never become glyphs in GNU `troff`. If not discarded (as when trailing text lines), horizontal motions represent them in the output.

In a `troff` system, a font description file (recall Section 2.4 [Font Directories], page 15) lists all of the glyphs a particular font provides. If the user requests a glyph not available in the currently selected font, the formatter looks it up an ordered list of *special fonts*. By default, the '`ps`' (PostScript) and '`pdf`' output devices support the two special fonts '`SS`' (slanted symbol) and '`S`' (symbol), these devices' `DESC` files arrange them such that the

[158] See Section 5.38 [Debugging], page 240.

[159] GNU `troff` looks up fonts by name with a search that is O(n) in the length n of the dynamically sized mounting position list.

formatter searches the former 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. GNU `troff`'s `special` and `fspecial` requests alter the list of fonts treated as special on a general basis, or only when a certain font is currently selected, respectively.

The formatter supports three kinds of character. An *ordinary character* is the most commonly used, has no special syntax, and typically represents itself.[160] Interpolate a *special character* with the '`\[xxx]`' or '`\C'xxx'`' escape sequence syntax, where *xxx* is an identifier. An *indexed character* bypasses most character-to-glyph resolution logic, uses the '`\N'i'`' syntax, and selects a glyph from the currently selected font by its integer-valued position *i* in the output device's representation of that font.[161]

User-defined characters are similar to string definitions,[162] and permit extension of or substitution within the character repertoire. Any ordinary, special, or indexed character can be user-defined. The `char`, `fchar`, `schar`, and `fschar` requests create user-defined characters employed at various stages of the character-to-glyph resolution process.

GNU `troff` employs the following procedure to resolve an input character into a glyph. User-defined characters make this resolution process recursive. The first step that succeeds ends the resolution procedure for the character being formatted, which may not be the last in the sequence interpolated by a user-defined character.

- Interpolate the definition of any character defined by the `char` request and apply this procedure to each character in its definition.
- Check the current font for a glyph corresponding to the character.
- Interpolate the definition of any user-defined character matching defined by the `fchar` request and apply this procedure to each character in its definition.
- Check whether the current font has a font-specific list of special fonts; if so, check the each font therein, in the order determined by the last applicable `fspecial` request, for a glyph corresponding to the character.
- Interpolate the definition of any character defined by the `fschar` request for the currently selected font, and apply this procedure to each character in its definition.
- Check each font in the list configured by the most recently issued `special` request for a glyph corresponding to the character.
- Interpolate the definition of any character defined by the `schar` request and apply this procedure to each character in its definition.

---

[160] Depending on the breadth of the output device's glyph repertoire, the characters `'`, `-`, `^`, `` ` ``, and `~` can be exceptions to this rule. `"` and `\` are not exceptions, but because they are syntactically meaningful to the formatter, access to their glyphs may require use of special characters (or changing or disabling the escape character). See *groff_char*(7).

[161] Fonts do not necessarily arrange their glyphs per a standard character encoding.

[162] See Section 5.22 [Strings], page 170.

- Finally, iterate through the list of mounted fonts by position; recall Section 5.19.3 [Font Mounting Positions], page 145. For each mounted font, if that font's description bears the `special` directive,[163] check it for a glyph corresponding to the character. This stage of the resolution process can sometimes lead to surprising results since the `fonts` directive in the `DESC` file often contains empty positions that are filled by a macro file or document employing the `fp` request after the formatter initializes.

  For example, consider the following `DESC` contents.

  ```
  fonts 3 0 0 ZD
  ```

  This directive mounts (only) font `ZD` at position 3. We assume that `ZD` is a special font, containing glyph `foo`, and that no other fonts have yet been mounted. The request

  ```
  .fspecial BAR BAZ
  ```

  makes font `BAZ` special while font `BAR` is selected. We further assume that `BAZ` is really a special font—that is, 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 mounting 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 the formatter starts the search for special fonts in the list of already mounted fonts, with increasing mounting positions. Consequently, it finds `BAZ` before `FOO` even before `XXX`, which is not the intended behaviour.

See Section 6.1 [Device and Font Description Files], page 257, and Section 5.19.6 [Special Fonts], page 155, for more details.

The *groff_char*(7) man page houses a complete list of predefined special character names, but the availability of any as a glyph is device- and font-dependent. For example, use the shell command

```
$ man -T dvi groff_char > groff_char.dvi
```

to obtain those available with the DVI device and default font configuration.[164] If you want to use an additional macro package to change the fonts used, you must run `groff` (or `troff`) directly.

```
$ groff -t -m ec -m an -T dvi groff_char.7 > groff_char.dvi
```

Except for those listed in *groff_char*(7), GNU `troff` derives special character names algorithmically, using a simplified version of the technique employed by the Adobe Glyph List (AGL) algorithm, described in `https://github.com/adobe-type-tools/agl-aglfn`. The (frozen) set of names that can't be derived algorithmically is called the *groff glyph list (GGL)*.

---

[163] See Section 6.1 [Device and Font Description Files], page 257.

[164] Not all versions of the `man` program support the `-T` option; use the subsequent example for an alternative.

- 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;[165] 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.
- GNU troff maintains a table to decompose all algorithmically derived glyph names that are composites. 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; GNU troff also automatically does the mapping. Example: The glyph u0045_0302 is mapped to ^E.
- glyph names of the GGL can't be used in composite glyph names; for example, ^E_u0301 is invalid.

\(*nm* [Escape sequence]
\[*name*] [Escape sequence]
\[*base-glyph combining-component* . . .] [Escape sequence]

Typeset a special character *name* (two-character name *nm*) or a composite glyph consisting of *base-glyph* overlaid with one or more *combining-components*. For example, '\[A ho]' is a capital letter "A" with a "hook accent" (ogonek).

There is no special syntax for one-character names—the analogous form '*n*' would collide with other escape sequences.

A special character name of length one is not the same thing as an ordinary character: that is, the character a is not the same as \[a]. If *name* is undefined, the formatter ignores the escape sequence.[166]

---

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

[166] GNU troff emits a warning in category 'char'. Plan 9 troff, uniquely, formats *name* as text.

GNU `troff` resolves `\[...]` with multiple components (arguments) as follows.

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

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

Examples:

`\[A ho]` '`A`' maps to `u0041`, '`ho`' maps to `u02DB`, thus the final glyph name would be `u0041_02DB`. This is not the expected result: the ogonek glyph '`ho`' is a spacing ogonek, but for a proper composite a non-spacing ogonek (U+0328) is necessary. Looking into the file `composite.tmac`, one can find '`.composite ho u0328`', which changes the mapping of '`ho`' while a composite glyph name is constructed, causing the final glyph name to be `u0041_0328`.

```
\[^E u0301]
\[^E aa]
\[E a^ aa]
\[E ^ ']
```

'`^E`' maps to `u0045_0302`, thus the final glyph name is `u0045_0302_0301` in all forms (assuming proper calls of the `composite` request).

It is not possible to define glyphs with names like '`A ho`' within a `groff` font file. This is not really a limitation; instead, you have to define `u0041_0328`.

`\C'`*xxx*`'` [Escape sequence]

Typeset the special character *xxx*. Normally, it is more convenient to use '`\[`*xxx*`]`', but `\C` has some advantages: it is compatible with AT&T device-independent `troff` and can interpolate special characters with '`]`' in their names. The delimiter need not be a neutral apostrophe; recall Section 5.6.5 [Delimiters], page 96.

`.composite` *c1 c2* [Request]

Map ordinary or special character name *c1* to *c2* when *c1* is a combining component in a composite character. See above for examples. This is a strict rewriting of the special character name; no check is performed for the existence of a glyph for either. Typically, `composite` is used to map a spacing character to a combining one. A set of default mappings for many accents can be found in the file `composite.tmac`, loaded by the default `troffrc` at startup.

Obtain a report of mappings defined by `composite` on the standard error stream with the `pcomposite` request.[167]

`\N'n'` [Escape sequence]

The `\N` *indexed character* escape sequence formats the glyph indexed *n* in the current font. *n* can be any non-negative decimal integer. If the current font lacks a glyph with that index, the formatter does *not* search special fonts. The formatter does not apply kerning to glyphs accessed with `\N`. The delimiter need not be a neutral apostrophe; recall Section 5.6.5 [Delimiters], page 96. *n* is not necessarily the input character code; use the Unicode form of special character escape sequence shown above to access glyphs by their character encoding. Most devices number glyphs with indices between 0 and 255 only; the `utf8` output device uses indices in the range 0–65535.

The `\N` escape sequence can be conveniently used in conjunction with the `char` request.

```
.char \[phone] \f[ZD]\N'37'
```

The code of each glyph is given in the fourth column in the font description file after the `charset` command. It is possible to include unnamed glyphs in the font description file by using a name of '`---`'; the `\N` escape sequence is the only way to use these.

A few escape sequences abbreviate special character escape sequences; these exist for syntactical or historical reasons.

`\'` [Escape sequence]

An escaped neutral apostrophe is a synonym for `\[aa]` (acute accent).

`\` ` [Escape sequence]

An escaped grave accent is a synonym for `\[ga]` (grave accent).

`\-` [Escape sequence]

An escaped hyphen-minus is a synonym for `\[-]` (minus sign).

`\_` [Escape sequence]

An escaped underscore ("low line") is a synonym for `\[ul]` (underrule). On typesetting devices, the underrule is font-invariant and drawn lower than the underscore '`_`'.

`.cflags` *n c...* [Request]

Assign properties encoded by non-negative integer *n* to each character or class[168] *c*. Spaces need not separate *c* arguments.

Characters, whether ordinary, special, or indexed, have certain associated properties. The argument *n* is the sum of the desired flags, and the

[167] See Section 5.38 [Debugging], page 240.

[168] See Section 5.19.5 [Character Classes], page 154.

remaining arguments are the characters to be assigned those properties. Some property combinations are nonsensical, such as '33' (1 + 32).

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

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

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

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

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

32 Mark the character as transparent for the purpose of end-of-sentence recognition. In other words, GNU troff treats an end-of-sentence character followed by any number of characters with this property as the end of a sentence if followed by a newline or two spaces.[169] Initially, characters '"')]*\[dg]\[dd]\[rq]\[cq]' have this property.

64 Ignore hyphenation codes of the surrounding characters. Use this value in combination with 2 and 4. Initially, no characters have this property.

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

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

into your document. However, this practice can lead to bad layout if done thoughtlessly; in most situations, a solution superior to altering a character's flags is inserting \: at the places that need a break point.

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

[169] This is the same as having a zero space factor in TEX.

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, GNU troff does not insert a break between them. If we use value 6 instead for the left character, a break after the character can't be suppressed since the neighboring character on the right doesn't get examined.

.char *c* ["][*contents*] [Request]
.fchar *c* ["][*contents*] [Request]
.fschar *f c* ["][*contents*] [Request]
.schar *c* ["][*contents*] [Request]

Define an ordinary, special, or indexed character *c* as *contents*. Omitting *contents* gives *c* an empty definition. GNU troff removes a leading neutral double quote '"' from *contents*, permitting initial embedded spaces in it, and reads it to the end of the input line in copy mode.[170]

Defining (or redefining) a character *c* creates a formatter object that GNU troff recognizes like any other ordinary, special, or indexed character on input, and produces *contents* on output. When formatting *c*, GNU troff processes *contents* in a temporary environment and encapsulates the result in a node,[171] disabling compatibility mode and treating \ as the escape character while interpreting *contents*. Any emboldening, constant spacing, or track kerning applies to this object rather than to individual glyphs resulting from the formatting of *contents*.

A character defined by these requests can be used just like a glyph provided by the output device. In particular, other characters can be translated to it with the tr and trin requests; it can be made the tab or leader fill character with the tc and lc requests, respectively; sequences of it can be drawn with the \l and \L escape sequences; it can be grouped into character classes;[172] and, if the hcode request is used on *c*, it is subject to automatic hyphenation.

---

[170] See Section 5.24.2 [Copy Mode], page 188.
[171] See Section 5.37 [GNU troff Internals], page 237.
[172] See Section 5.19.5 [Character Classes], page 154.

However, a user-defined character *c* does not participate at its boundaries in kerning adjustments or italic corrections.

GNU `troff` prevents infinite recursion by treating an occurrence of a character in its own definition as if it had no user-directed definition; if a component character remains undefined at the time it is interpolated, it does not format.[173] Mutually recursive character definitions are handled similarly.

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

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

`.rchar` *c* ... [Request]
`.rfschar` *f c* ... [Request]

Remove definition of each ordinary, special, or indexed character *c*, undoing the effect of a `char`, `fchar`, or `schar` request. Spaces need not separate *c* arguments. The character definition removed (if any) is the first encountered in the resolution process documented above. Glyphs, which are defined by font description files, cannot be removed.

`rfschar` removes character definitions created by `fschar` for font *f*.

5.19.5 Character Classes

GNU `troff` can group characters into *classes*, making manipulation of their breaking and/or sentential properties convenient; recall the `cflags` request

[173] GNU `troff` emits a warning in category '`char`'. See Section 5.38.1 [Warnings], page 245.

in Section 5.19.4 [Characters and Glyphs], page 146. Classes are particularly useful for East Asian languages such as Chinese, Japanese, and Korean, which have much larger character repertoires than the Latin, Greek, Cyrillic, or Thai scripts. In such large character sets, many characters share the same properties. Only `class` and `cflags` requests can operate on character classes.

`.class` *ident c* . . . [Request]

Define a character class (or simply "class") *ident* comprising the members *c* . . ., where each *c* is an ordinary, special, or indexed character; or a *range expression*. A class thus defined can then be referred to in a `cflags` request in lieu of listing all the characters within it.

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

Since class and special character names share the same name space, we recommend starting and ending the class name with '`[`' and '`]`', respectively, to avoid collisions with existing special 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 '`\[...]`', you must therefore access a class thus named via the `\C` escape sequence.

An argument *c* can alternatively be a *range expression* 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 determine the start and end values of the range. If that conversion fails, GNU `troff` ignores the range expression and any remaining arguments.

If you want to include '`-`' in a class, it must be the first character in a *c* argument; otherwise GNU `troff` interprets the argument as a range expression.

5.19.6 Special Fonts

Special fonts are those that the formatter searches, in mounting position order, when it cannot find a requested glyph in the selected font. Typically, they are declared as such in their description files,[174] and contain unstyled glyphs. The "Symbol" and "Zapf Dingbats" fonts of the PostScript and PDF standards are examples. Ordinarily, only typesetters have special fonts.

GNU `troff`'s `special` and `fspecial` requests permit a document to supplement the set of fonts the device configures for glyph search without having to use the `fp` request to manipulate the list of mounting positions, which can be tedious—by default, GNU `troff` mounts 40 fonts at startup when using the `ps` device.

[174] See Section 6.1.2 [Font Description File Format], page 260.

`.special` [*s* . . .] [Request]
`.fspecial` *f* [*s* . . .] [Request]

`special` declares each font *s* as special, irrespective of its description file, populating a list that GNU `troff` searches, in order, to find the glyph demanded. GNU `troff` mounts each font *s*, if it exists. Invoking `special` without arguments empties the list. A font is not automatically unmounted if a subsequent `special` request removes it from the list. Initially, the list is empty.

`fspecial` is similar; it designates each font *s* as special only when font *f* is selected. Initially, a font *f*'s list of associated special fonts is empty for all *f*.

5.19.7 Font Effects and Text Underlining

Some requests and escape sequences apply simple transformations to glyph shapes, or draw rules beneath the text baseline (underlining). These features are largely vestiges of the days when output devices lacked a wide variety of fonts, and when `nroff` and `troff` were separate formatters. GNU `troff` retains support for them even though they are seldom needed nowadays.

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

Set (increment, decrement) the height of the current font, but not its width. If *height* is zero, the formatter uses the font's inherent height for its type size. The default scaling unit is '`z`'.

Changing the font height does not affect vertical spacing; dramatic changes may be better accompanied by an `\x` escape sequence to add extra pre-vertical space to the output line. Recall Section 5.11 [Manipulating Spacing], page 122.

The read-only register `.height` interpolates the font height.

As of this writing, only the `ps` and `pdf` output devices support this feature.

The formatter does not tokenize `\H` when reading it; the escape sequence updates the environment.[175] It thus can be used in requests that expect a single-character argument. We can alter the font height of a margin character as follows.[176]

```
.mc \H'+5z'x\H'0'
```

In compatibility mode,[177] GNU `troff` behaves differently: it applies an increment or decrement to the current type size and not to the previously selected font height.

[175] See Section 5.32 [Environments], page 224.
[176] See Section 5.26 [Output Line Annotation], page 197.
[177] See Section 5.39.2 [Compatibility Mode], page 248.

```
.cp 1
\H'+5'test \H'+5'test
```

prints the word '`test`' twice with the same font height—five points larger than the current font size.

`\S'`*slant*`'` [Escape sequence]
`\n[.slant]` [Register]

Slant the glyphs of the currently selected font by *slant* degrees. Positive values slant in the direction of text flow. Only integer values are possible.

The read-only register `.slant` interpolates the font slant.

As of this writing, only the `ps` and `pdf` output devices support this feature.

The formatter does not tokenize `\S` when reading it; the escape sequence updates the environment.[178] It thus can be used in requests that expect a single-character argument. We can apply a slant to a margin character as follows.[179]

```
.mc \S'20'x\S'0'
```

This escape sequence is incorrectly documented in the AT&T `troff` manual: the slant is only assigned, never incremented or decremented.

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

Underline (in `nroff` mode) or italicize (in `troff` mode; recall Section 5.14 [`troff` and `nroff` Modes], page 130) the next *n* productive input lines. A nonpositive argument cancels any pending underlining. Omitting the argument implies an *n* of '`1`'. The read-only register `.ul` stores the count of productive input lines remaining to be underlined, which is associated with the environment.[180]

When `ul` commences underlining, the formatter stores the currently selected font (strictly, its mounting position), and restores that font when underlining ceases. Use the `uf` request to select an alternative underline font.

The `ul` request does not cause spaces to be underlined.

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

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

`.uf` *font* [Request]

Set the underline font (globally) used by `ul` and `cu`. By default, this is the font at mounting position 2. *font* is a font identifier or mounting position.

---

[178] See Section 5.32 [Environments], page 224.
[179] See Section 5.26 [Output Line Annotation], page 197.
[180] See Section 5.32 [Environments], page 224.

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

Embolden *font* by overstriking its glyphs offset by *offset* units minus one.

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 is a font identifier or mounting position.

  offset is available in the `.b` read-only register if a special font is active; in the `bd` request, its default unit is '`u`'.

- Imitate a bold form conditionally. Embolden *font1* by *offset* only if font *font2* is the current font. This request can be issued repeatedly to set up different emboldening values for different current fonts. If the second argument is missing, emboldening is turned off for this particular current font.

  Because the emboldening is conditional, it applies only if the glyph to be formatted is not available in the current font. *font1* must therefore be a special font, configured either with the `special` directive in its font description file or with the `fspecial` request).

`.cs` *font* [*width* [*em-size*]] [Request]

Apply *constant glyph spacing* to font *font*. The *advance width* of every glyph in the font—the amount the drawing position moves horizontally after setting the glyph—becomes *width*/36 ems. The *em-size* argument overrides the font's em size; omitting it preserves the formatter's usual recomputation of *font*'s em size from the type size in effect each time that font is selected. Omitting both the second and third arguments deactivates constant glyph spacing for font *font*.

The default scaling unit for *em-size* is '`z`'; *width* is an integer.

5.19.8 Ligatures and Kerning

Proportional fonts commonly employ two techniques to improve the esthetics of typeset text. *Ligatures* are sequences of glyphs that are visually connected or "tied", overlapping them and slightly altering their shapes. *Kerning* is the adjustment of horizontal spacing between glyphs. Neither is employed on terminals.[181]

Most typesetters support ligatures for the sequences 'fi', 'fl', 'ff', 'ffi', and 'ffl', and `troff` formatters do likewise. Some fonts may include others, but GNU `troff` does not (yet) support them.

[181] A monospaced font may possess ligature glyphs, but they nevertheless seldom see use to set text.

The formatter checks only the current font for ligatures and kerning adjustments: neither glyphs from special fonts nor special characters defined with the `char` request (and its siblings) are considered for these processes; recall Section 5.19.4 [Characters and Glyphs], page 146.

`.lg` [*flag*] [Request]
`\n[.lg]` [Register]

Switch the ligature mechanism on or off; if the parameter is non-zero or missing, ligatures are enabled, otherwise disabled. Default is on. The current ligature mode can be found in the read-only register `.lg` (set to 1 or 2 if ligatures are enabled, 0 otherwise).

Setting the ligature mode to 2 enables the two-character ligatures (fi, fl, and ff) and disables the three-character ligatures (ffi and ffl).

Pairwise kerning modifies the distance between adjacent glyphs in a pair, improving readability. In most cases, this alteration is a decrease. Consider the letters 'V' and 'A' set adjacently. With kerning, 'VA' is rendered. Without kerning, they appear as 'VA'. Monospaced (typewriter-like) fonts, and therefore terminals, don't use kerning.

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

Enable or disable pairwise kerning of glyphs in the environment per *b*. It is enabled by default, and if *b* is omitted.

The read-only register `.kern` interpolates 1 if pairwise kerning is enabled, 0 otherwise.

If the font description file contains pairwise kerning information, GNU `troff` kerns glyphs from that font.[182] Kerning between two glyphs can be inhibited by placing `\&` between them: '`V\&A`'.

Track kerning expands or reduces the space between glyphs. This can be handy, for example, if you need to squeeze a long word onto a single line or spread some text to fill a narrow column. It must be used with great care since it is usually considered bad typography if the reader notices the effect.

`.tkf` *f s1 n1 s2 n2* [Request]

Enable track kerning for font *f*. If the selected font is *f*, increase the advance width by an amount between *n1* and *n2* (each can be negative): if the type size is less than or equal to *s1*, or if *s2* is less than or equal to *s1*, the width increases by *n1*; if the type size is greater than or equal to *s2*, the width increases by *n2*; otherwise, the advance width increase is a linear function of the type size.

The default scaling unit is '`z`' for *s1* and *s2*, and '`p`' for *n1* and *n2*. GNU `troff` adds the track kerning amount even to the rightmost glyph in a line; for large *n1* and/or *n2*, we thus recommend increasing the line length to compensate.

[182] See Section 6.1.2 [Font Description File Format], page 260.

5.19.9 Italic Corrections

When typesetting adjacent glyphs from typefaces of different slants, the space between them may require adjustment.

`\/` [Escape sequence]

Apply an *italic correction*: modify the spacing of the preceding glyph so that the distance between it and the following glyph is correct if the latter is of upright shape. For example, if an italic '`f`' is followed immediately by a roman right parenthesis, then in many fonts the top right portion of the '`f`' overlaps the top of the right parenthesis, which is ugly. Use `\/` whenever a slanted glyph is followed immediately by an upright glyph without any intervening space.

`\,` [Escape sequence]

Apply a *left italic correction*: modify the spacing of the following glyph so that the distance between it and the preceding glyph is correct if the latter is of upright shape. For example, if a roman left parenthesis is immediately followed by an italic '`f`', then in many fonts the bottom left portion of the '`f`' overlaps the bottom of the left parenthesis, which is ugly. Use `\,` whenever an upright glyph is followed immediately by a slanted glyph without any intervening space.

5.19.10 Dummy Characters

As discussed in Section 5.1.7 [Requests and Macros], page 71, the formatter treats the first character on an input line specially. Further, formatting a glyph has many consequences on formatter state.[183] Occasionally, we want to escape this context or embrace some of those consequences without actually rendering a glyph to the output.

`\&` [Escape sequence]

Interpolate a dummy character, which is constitutive of output but invisible.[184] Its presence alters the interpretation context of a subsequent input character, and enjoys several applications.

---

[183] See Section 5.32 [Environments], page 224.

[184] Opinions of this escape sequence's best name abound. "Zero-width space" is a popular misnomer: `roff` formatters do not treat it like a space; when filling, they do not break a line where `\&` appears. Ossanna called it a "non-printing, zero-width character", but the character causes *output* even though it does not "print". If no output line is pending, the dummy character starts one. Contrast an empty input document with one containing only `\&`. The former produces no output; the latter, a blank page.

- Prevent insertion of supplemental inter-sentence space after an end-of-sentence character.

```
Test.
Test.
.br
************
    ⇒ Test.  Test.
    ⇒ ************
Test.\&
Test.
.br
************
    ⇒ Test. Test.
    ⇒ ************
```

- Prevent recognition of a control character.

```
.Test
    [error]  warning: name 'Test' not defined
\&.Test
    ⇒ .Test
```

- Prevent kerning between two glyphs.

```
VAVAVA
    ⇒ VAVAVA
V\&A\&V\&A\&V\&A
    ⇒ VAVAVA
```

- Translate a character to "nothing".

```
.tr JIjiK\&k\&UVuv
Post universitum, alea jacta est, OK?
    ⇒ Post vniversitvm, alea iacta est, O?
```

- Mark the end of a numerical expression in an otherwise ambiguous context.

```
\l'4i-'
    [error]  error: numeric expression containing
    [error]  character "'" is invalid
\l'4i\&-'
    ⇒ ----------------------------------------
```

The dummy character escape sequence sees use in macro definitions as a means of ensuring that arguments are treated as text even if they begin with spaces or control characters.

```
.de HD \" typeset a simple bold heading
.  sp
.  ft B
\&\\$1 \" exercise: remove the \&
.  ft
.  sp
..
.HD .\|.\|.\|surprised?
```

One way to think about the dummy character is to imagine placing the symbol '`&`' in the input at a certain location; if doing so has all the side effects on formatting that you desire except for sticking an ugly ampersand in the midst of your text, the dummy character is what you want in its place.

`\)` [Escape sequence]

Interpolate a *transparent* dummy character—one that is transparent to end-of-sentence detection. It behaves as `\&`, except that `\&` is treated as letters and numerals normally are after '`.`', '`?`' and '`!`'; `\&` cancels end-of-sentence detection, and `\)` does not.

```
.de Suffix-&
.  nop \&\\$1
..
.
.de Suffix-)
.  nop \)\\$1
..
.
Here's a sentence.\c
.Suffix-& '
Another one.\c
.Suffix-) '
And a third.
**********************************************
    ⇒ Here's a sentence.' Another one.'  And a third.
    ⇒ **********************************************
```

5.20 Manipulating Type Size and Vertical Spacing

These concepts were introduced in Section 5.2 [Page Geometry], page 79. The height of a font's tallest glyph is one em, which is equal to the type size in points.[185] A vertical spacing of less than 120% of the type size can make a document hard to read. Larger proportions can be useful to spread the text for annotations or proofreader's marks. By default, GNU troff uses 10 point type on 12 point spacing. Typographers call the difference between type size and vertical spacing *leading*.[186] Both properties are associated with the environment.[187]

5.20.1 Changing the Type Size

.ps [*size*] [Request]
.ps +*size* [Request]
.ps −*size* [Request]
\n[.s] [Register]

Set the type size to, or alter it by, *size*. ps with no argument restores the previous size. The ps request's default scaling unit is 'z'; recall Section 5.3 [Measurements], page 80, and see Section 5.20.3 [Using Fractional Type Sizes], page 166). The formatter rounds the requested size to the nearest valid size (with ties rounding down) within the limits supported by the device, and if the requested size is non-positive, treats it as 1 u.

Type size alteration is incorrectly documented in the AT&T troff manual, which claims "if [the requested size] is invalid, the next larger valid size will result, with a maximum of 36".[188]

The read-only string-valued register .s interpolates the type size in points as a decimal fraction. To obtain the type size in scaled points, interpolate the .ps register instead.[189]

\s*size* [Escape sequence]

The \s escape sequence also determines the type size, but handles a zero argument differently. It supports a variety of syntax forms.

\s*n* Set the type size to *n* typographical points. *n* must be a single digit.[190] If *n* is '0', restore the previous size.

---

185 In text fonts, parentheses are often the tallest glyphs, but a font's glyphs may not match the nominal type size! In the standard PostScript font families, 10-point Times sets better with 9-point Helvetica and 11-point Courier than if all were used at 10 points. Recall the fzoom request in Section 5.19.1 [Selecting Fonts], page 141, for a remedy.

186 Rhyme with "sledding"; mechanical typography used lead metal (Latin *plumbum*).

187 See Section 5.32 [Environments], page 224.

188 The claim appears to have been true of Ossanna troff for the C/A/T device; Kernighan made device-independent troff more flexible.

189 See Section 5.20.3 [Using Fractional Type Sizes], page 166.

190 In compatibility mode only, a non-zero *n* must be in the range 4–39. See Section 5.39.2 [Compatibility Mode], page 248.

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

\s(*nn* Set the type size to *nn* typographical points. *nn* must be exactly two digits. If *n* is '00', restore the previous size.

\s+(*nn*
\s-(*nn*
\s(+*nn*
\s(-*nn* Alter the type size in scaled points by the *nn* typographical points. *nn* must be exactly two digits.

Section 5.20.3 [Using Fractional Type Sizes], page 166, presents further syntactical forms of the \s escape sequence that additionally accept decimal fractions.

```
snap, snap,
.ps +2
grin, grin,
.ps +2
wink, wink, \s+2nudge, nudge,\s+8 say no more!
.ps 10
```

The formatter does not tokenize \s when reading it; the escape sequence updates the environment.[191] It thus can be used in requests that expect a single-character argument. We can alter the type size of a margin character as follows.[192]

```
.mc \s[20]x\s[0]
```

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

Set the available type sizes GNU troff permits.[193] Arguments are in scaled points.[194] Each can be a single type size (such as '12000'), or a range of sizes (such as '4000-72000'). You can optionally end the list with a '0'.

5.20.2 Changing the Vertical Spacing

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

Set the vertical spacing to, or alter it by, *space*. The default scaling unit is 'p'. If *space* is omitted, the formatter resets the vertical spacing to its

[191] See Section 5.32 [Environments], page 224.
[192] See Section 5.26 [Output Line Annotation], page 197.
[193] The output device's DESC file specifies which type sizes are normally allowed; see Section 6.1.1 [DESC File Format], page 257.
[194] See Section 5.20.3 [Using Fractional Type Sizes], page 166.

previous value, or the default if there is none. If *space* is negative, the formatter sets the vertical spacing to the smallest possible positive value, the vertical motion quantum (as found in the .V register).[195]

'.vs 0' isn't saved in a diversion since it doesn't result in a vertical motion.[196] You must explicitly issue this request before interpolating the diversion if you desire its effect.

The read-only register .v interpolates the vertical spacing.

When a break occurs, GNU troff performs the following procedure.

- Move the drawing position vertically by the *extra pre-vertical line space*, the minimum of all negative \x escape sequence arguments in the pending output line. If this motion springs a vertical position trap,[197] including the implicit page trap,[198] interrupt this procedure.
- Move the drawing position vertically by the vertical line spacing. If this motion springs a trap, including the implicit page trap, interrupt this procedure.
- An interrupted break procedure resumes here after traps are handled. Apply any configured adjustment and marginal annotations,[199] and append the pending output line to the current diversion[200]. (Often, the "diversion" corresponds to the the output device's current page.)
- Move the drawing position vertically by the *extra post-vertical line space*, the maximum of all positive \x escape sequence arguments in the line that has just been output. If this motion springs a trap, including the implicit page trap, abort this procedure.
- Move the drawing position vertically by the *post-vertical line spacing* (see below). If this motion springs a trap, including the implicit page trap, abort this procedure.

If the break procedure is interrupted or aborted because a vertical position trap is sprung, GNU troff updates the values of the .ne and .trunc registers.[201] Encountering the implicit page trap (*ibid.*) does not alter them.

Prefer vs or pvs over ls to produce double-spaced documents. vs and pvs have finer granularity than ls; moreover, some preprocessors assume single spacing. Recall Section 5.11 [Manipulating Spacing], page 122, regarding the \x escape sequence and the ls request.

---

[195] GNU troff emits a warning in category 'range'. See Section 5.38.1 [Warnings], page 245.

[196] See Section 5.30 [Diversions], page 217.

[197] See Section 5.29.1 [Vertical Position Traps], page 206.

[198] See Section 5.29.1.2 [The Implicit Page Trap], page 211.

[199] See Section 5.26 [Output Line Annotation], page 197.

[200] See Section 5.30 [Diversions], page 217.

[201] See Section 5.29.1.1 [Page Location Traps], page 206.

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

Set the post-vertical spacing to, or alter it by, *space*. The default scaling unit is '`p`'. If `pvs` is invoked without an argument, the post-vertical spacing is reset to the previous value before the last call to `pvs`. GNU `troff` emits a warning in category '`range`' if *space* is negative; the post-vertical spacing is then set to zero.

The read-only register `.pvs` interpolates the post-vertical spacing.

5.20.3 Using Fractional Type Sizes

When configuring the type size, AT&T `troff` ignored scaling units and interpreted all measurements in points. Combined with integer arithmetic, this design choice made it impossible to support, for instance, ten-and-a-half-point type. In GNU `troff`, an output device can select a scaling factor that subdivides a point into "scaled points". A type size expressed in scaled points can thus represent a non-integral size in points.

A *scaled point*, scaling unit `s`, is equal to 1/*sizescale* points, where the device description file, `DESC`, specifies *sizescale* and otherwise defaults to 1.[202] GNU `troff` also defines the *typographical point*, scaling unit `z`, which explicitly specifies a type size of potentially non-integral measure. The program multiplies typographical points by *sizescale* and converts the value to an integer. Arguments GNU `troff` interprets in `z` units by default comprise those to the escape sequences `\H` and `\s`, to the request `ps`, the third argument to the `cs` request, and the second and fourth arguments to the `tkf` request.

For example, if *sizescale* is 1000, then a scaled point is one thousandth of a point. The request '`.ps 10.5`' is synonymous with '`.ps 10.5z`'; both set the type size to 10,500 scaled points, or 10.5 typographical points.

`\n[.ps]` [Register]

This read-only register interpolates the type size in scaled points. '`\n[.ps]s`', '`\n[.s]z`', and '`1m`' are co-equal by definition.

[202] See Section 6.1 [Device and Font Description Files], page 257.

```
.tm device=\*[.T]
.tm A: .s=\n[.s]z, .ps=\n[.ps]s
.ps 10.5
.tm B: .s=\n[.s]z, .ps=\n[.ps]s
.ps 12.3p
.tm C: .s=\n[.s]z, .ps=\n[.ps]s
.ps 8.1z
.tm D: .s=\n[.s]z, .ps=\n[.ps]s
.ps 10500s
.tm E: .s=\n[.s]z, .ps=\n[.ps]s
    ⇒ device=ps
    ⇒ A: .s=10z, .ps=10000s
    ⇒ B: .s=10.5z, .ps=10500s
    ⇒ C: .s=12.3z, .ps=12300s
    ⇒ D: .s=8.1z, .ps=8100s
    ⇒ E: .s=10.5z, .ps=10500s
```

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

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

Output devices may be limited in the type sizes they can employ. The .s and .ps registers represent the type size selected by the formatter as it understands a device's capability. The read-only registers .psr and (string-valued) .sr interpolate the last *requested* type size in scaled and typographical points, respectively. Like the current and previous selected type size, the requested ones are properties of an environment.[203]

For example, if a document requests a type size of 10.95 points, and the nearest size permitted by a sizes request (or by the sizes or sizescale directives in the device's DESC file) is 11 points, groff uses the latter value.

The \s escape sequence offers the following syntax forms that work with fractional type sizes and accept scaling units. The delimited forms need not use the neutral apostrophe; see Section 5.6.5 [Delimiters], page 96.

\s[*n*]
\s'*n*' Set the type size to *n* typographical points; *n* is a numeric expression with a default scaling unit of 'z'.

[203] See Section 5.32 [Environments], page 224.

\s[+*n*]
\s[-*n*]
\s+[*n*]
\s-[*n*]
\s'+*n*'
\s'-*n*'
\s+'*n*'
\s-'*n*'

Increase or decrease the type size by *n* typographical points; *n* is a numeric expression with a default scaling unit of 'z'. If *n* is '0', restore the previous size.

5.21 Colors

GNU troff supports color output with a variety of color spaces and up to 16 bits per channel. Some devices, particularly terminals, may be more limited. When color support is enabled, two colors are current at any given time: the *stroke color*, with which glyphs, rules (lines), and geometric figures are drawn, and the *fill color*, which paints the interior of filled geometric figure.

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

Enable or disable output of color-related device-independent output commands per Boolean expression *b*. It is enabled by default, and if *b* is omitted.

The read-only register .color interpolates 1 if color support is enabled, 0 otherwise.

Color can also be disabled with the -c command-line option.

.defcolor *ident scheme color-component* . . . [Request]

Define a color named *ident*. *scheme* selects a color space and determines the quantity of required *color-component*s; it must be one of 'rgb' (three components), 'cmy' (three), 'cmyk' (four), or 'gray' (one). 'grey' is accepted as a synonym of 'gray'. The color components can be encoded as a single hexadecimal value starting with '#' or '##'. The former indicates that each component is in the range 0–255 (0–FF), the latter the range 0–65,535 (0–FFFF).

```
.defcolor half gray #7f
.defcolor pink rgb #FFC0CB
.defcolor magenta rgb  ##ffff0000ffff
```

Alternatively, each color component can be specified as a decimal fraction in the range 0–1, interpreted using a default scaling unit of f, which multiplies its value by 65,536 (but clamps it at 65,535).

```
.defcolor gray50 rgb 0.5 0.5 0.5
.defcolor darkgreen rgb 0.1f 0.5f 0.2f
```

Obtain a report of colors defined by `defcolor` on the standard error stream with the `pcolor` request.[204]

Each output device has a color named '`default`', which cannot be redefined. A device's default stroke and fill colors are not necessarily the same. For the `dvi`, `html`, `pdf`, `ps`, and `xhtml` output devices, GNU `troff` automatically loads a macro file defining many color names at startup. By the same mechanism, the devices supported by `grotty` recognize the eight standard ISO 6429/ECMA-48 color names.[205]

`.gcolor` [*col*] [Request]
`\m`*c* [Escape sequence]
`\m(`*co* [Escape sequence]
`\m[`*col*`]` [Escape sequence]
`\n[.m]` [Register]

Select *col* as the stroke color for glyphs, rules, and objects drawn with `\D'...'` escape sequences. The escape sequence `\M[]` restores the previous stroke color, or the default if there is none, as does a `gcolor` request without an argument.

```
.gcolor red
The next words
.gcolor
\m[red]are in red\m[]
and these words are in the previous color.
```

The stroke color selection is available in the read-only string-valued register '`.m`'. The default stroke color is named '`default`'.

GNU `troff` does not tokenize `\m` when reading it; the escape sequence updates the environment.[206] It thus can be used in requests that expect a single-character argument. We can assign a stroke color to a margin character as follows.[207]

```
.mc \m[red]x\m[]
```

`.fcolor` [*col*] [Request]
`\M`*c* [Escape sequence]
`\M(`*co* [Escape sequence]
`\M[`*col*`]` [Escape sequence]
`\n[.M]` [Register]

Select *col* as the fill color for objects drawn with `\D'...'` escape sequences. The escape sequence `\M[]` restores the previous fill color, or the default if there is none, as does an `fcolor` request without an argument.

Create an ellipse with a red interior as follows.

```
\M[red]\h'0.5i'\D'E 2i 1i'\M[]
```

[204] See Section 5.38 [Debugging], page 240.
[205] These are known vulgarly as "ANSI" colors, after its X3.64 standard, now withdrawn.
[206] See Section 5.32 [Environments], page 224.
[207] See Section 5.26 [Output Line Annotation], page 197.

The current environment's fill color selection is available in the read-only string-valued register '.M'.[208] The default fill color is named 'default'.

GNU troff does not tokenize \M when reading it; the escape sequence updates the environment.[209] It thus can be used in requests that expect a single-character argument. We can assign a fill color to a margin character as follows;[210] grotty interprets the fill color as a character cell background color.

```
.mc \m[black]\M[green]x\M[]\m[]
```

5.22 Strings

GNU troff supports strings primarily for user convenience. Conventionally, if one would define a macro only to interpolate a small amount of text, without invoking requests or calling any other macros, one defines a string instead. The language prefedines only one string.

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

The ds request creates a string with a specified name and contents, as appends to a string, and the * escape sequence dereferences a string, interpolating its contents. If the string named by the * escape sequence does not exist, the formatter defines it as empty and interpolates nothing.[211]

.ds *name* [["]*contents*] [Request]
.ds1 *name* [["]*contents*] [Request]
**n* [Escape sequence]
*(*nm* [Escape sequence]
*[*name* [*arg1 arg2* . . .]] [Escape sequence]
Define a string called *name* with contents *contents*. In GNU troff, if *name* already exists as an alias, the request replaces the contents of the alias's target; see als and rm below. Omitting *contents* defines *name* as an empty string. Otherwise, the formatter reads *contents* in copy mode. * is itself interpreted even in copy mode.[212]

The * escape sequence interpolates a previously defined string *name* (one-character name *n*, two-character name *nm*). Its bracketed interpolation form accepts arguments that are handled as macro arguments are; recall Section 5.6.3 [Calling Macros], page 92. In contrast to macro calls, however, if a closing bracket '] ' occurs in a string argument, that argument must be enclosed in double quotes. When defining strings, argument

[208] See Section 5.32 [Environments], page 224.
[209] See Section 5.32 [Environments], page 224.
[210] See Section 5.26 [Output Line Annotation], page 197.
[211] GNU troff emits a warning in category 'mac'. See Section 5.38.1 [Warnings], page 245.
[212] See Section 5.24.2 [Copy Mode], page 188.

interpolations must be escaped if they are to reference parameters from the calling context.[213]

```
.ds cite (\\$1, \\$2)
Gray codes are explored in \*[cite Morgan 1998].
    ⇒ Gray codes are explored in (Morgan, 1998).
```

Caution: After the formatter has read the space character that ends the first argument, it treats the remainder of the input line as the second argument, including any spaces, up to a newline or comment escape sequence. Ending string definitions (and appendments) with a comment, even an empty one, prevents unwanted space from creeping into them during source document maintenance.

```
.ds Si silicon \" use chemical symbol
My tricorder indicates a \*[Si]-based life form.
    ⇒ My tricorder indicates a silicon -based life form.
```

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

```
.ds Si silicon\" use chemical symbol
My tricorder indicates a \*[Si]-based life form.
    ⇒ My tricorder indicates a silicon-based life form.
```

Because the first space after the string name separates the arguments, you can retain it while using a comment to document an empty string.

```
.ds author Alice Pleasance Liddell\"
.ds friends \" empty; append to with .as
```

The formatter removes a leading neutral double quote '`"`' from *contents*, permitting initial embedded spaces in it. It interprets any other '`"`' literally, but the wise author uses the special character escape sequence `\[dq]` instead if the string might be interpolated as part of a macro argument; recall Section 5.6.3 [Calling Macros], page 92.

```
.ds salutation "         Yours in a white wine sauce,\"
.ds c-var-defn "  char mydate[]=\[dq]2020-07-29\[dq];\"
```

Strings are not limited to a single input line of text. `\RET` works just as it does elsewhere. The resulting string is stored *without* the newlines. When filling is disabled, care is required to avoid oversetting the line when interpolating strings.

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

Conversely, when filling is enabled, it is not necessary to append `\c` to a string interpolation to prevent a break afterward, as might be required in a macro argument. Nor does a string require use of GNU `troff`'s `chop` request to excise a trailing newline as is often done with diversions.[214]

---

[213] See Section 5.24.1 [Parameters], page 186.

[214] See Section 5.30 [Diversions], page 217.

It is not possible to embed a newline in a string that the formatter interprets as such when interpolating it. To achieve that effect, use `\*` to interpolate a macro instead.[215]

Because strings are similar to macros, they too can be defined so as to suppress AT&T `troff` compatibility mode when used.[216] The `ds1` request defines a string such that compatibility mode is off when the string is later interpolated. To be more precise, GNU `troff` inserts a *compatibility save* token at the beginning of *contents*, and a *compatibility restore* token at the end.[217]

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

`.as` *name* [["]*contents*] [Request]
`.as1` *name* [["]*contents*] [Request]

The `as` request is similar to `ds`, but appends *contents* to the string stored as *name* instead of repopulating it. If *name* does not already exist, the formatter creates it. Omitting *contents* performs no operation, beyond dereferencing (and thus possibly creating) the string.

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

Caution: The formatter reads the second argument to the end of the line in copy mode, omitting any leading neutral double quote '`"`' character. See the discussion of the `ds` request above.

The `as1` request works as does `as`, but like `ds1`, it brackets *contents* with *compatibility save* and *restore* tokens.

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* [["]*contents*] [Request]

Compute the number of characters in *contents* and store the count in the register *reg*. If *reg* doesn't exist, GNU `troff` creates it.

---

[215] See Section 5.31 [Punning Names], page 222.
[216] See Section 5.24 [Writing Macros], page 182.
[217] See Section 5.39.2 [Compatibility Mode], page 248.

GNU troff removes a leading neutral double quote '"' from *contents*, permitting initial embedded spaces in it, and reads it to the end of the input line in copy mode.[218]

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

Caution: The formatter reads the second argument to the end of the line in copy mode, omitting any leading neutral double quote '"' character. See the discussion of the ds request above.

Caution: If you interpolate a macro or diversion in *contents*,[219] the length request counts characters (or nodes) only up to the first newline, and leaves the rest on the input stream. In conventional circumstances, that means the remainder is interpreted, and may be formatted. To discover the length of any string, macro, or diversion, use the pm request.[220]

.chop *object* [Request]

Remove the last character from the macro, string, or diversion named *object*. This is useful for removing the newline from the end of a diversion that is to be interpolated as a string. You can apply chop repeatedly to the same *object*.[221]

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

Replace the string named *str* with its substring bounded by the indices *start* and *end*, inclusively. The first element of the string has index 0. Omitting *end* implies the largest valid value (the string length minus one). Negative indices count backward from the end of the string: the last element has index −1, the element before the last has index −2, and so on. If both *start* and *end* lie outside the string, GNU troff ignores the request.[222]

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

[218] See Section 5.24.2 [Copy Mode], page 188.
[219] See Section 5.31 [Punning Names], page 222.
[220] See Section 5.38 [Debugging], page 240.
[221] See Section 5.37 [GNU troff Internals], page 237, for details of nodes inserted automatically by GNU troff.
[222] It also emits a warning in category 'range'. See Section 5.38.1 [Warnings], page 245.

Compatibility save and restore tokens do *not* count for the purpose of index computation, even though they appear in string contents reported by the `pm` request and figure in the value computed by the `length` request.

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

Alter the string named *str* by replacing each of its bytes with its lowercase (`stringdown`) or uppercase (`stringup`) version (if one exists). GNU `troff`'s built-in special characters often transform in the expected way due to the regular naming convention for accented characters.[223] When they do not, use substrings and/or catenation.

```
.ds resume R\['e]sum\['e]\"
\*[resume]
.stringdown resume
\*[resume]
.stringup resume
\*[resume]
    ⇒ Résumé résumé RÉSUMÉ
```

`.rn` *old new* [Request]

Rename the request, macro, diversion, or string *old* to *new*.

`.rm` *name* . . . [Request]

Remove each request, macro, diversion, or string *name*. GNU `troff` treats subsequent interpolations or calls of *name* as if it had never been defined. The formatter performs no operation on a nonexistent *name*. If *name* is an invalid identifier, GNU `troff` diagnoses an error, but processes any subsequent valid *name* arguments.

This request is incorrectly documented in the AT&T `troff` manual as accepting only one argument.

`.als` *new-name existing-name* [Request]

Create alias (additional name) *new-name* of request, string, macro, or diversion *existing-name*, causing the names to refer to the same stored object. If *existing-name* is undefined, the formatter ignores the request.[224] If *new-name* already exists, its contents are lost unless already aliased.

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.

---

[223] See *groff_char*(7).

[224] GNU `troff` emits a warning in category '`mac`'. See Section 5.38.1 [Warnings], page 245.

Now consider this example.

```
.de foo
..
.
.als bar foo
.
.de bar
.  foo
..
.
.bar
```

```
error  input stack limit exceeded (probable infinite
error  loop)
```

In the above, `bar` remains an *alias*—another name for—the object referred to by `foo`, which the second `de` request replaces. Alternatively, imagine that the `de` request *dereferences* its argument before replacing it. Either way, the result of calling `bar` is a recursive loop that finally leads to an error.[225]

To remove an alias, call `rm` on its name. The object itself is not destroyed until it has no more names.

When a request, macro, string, or diversion is aliased, redefinitions and appendments "write through" alias names. To replace an alias with a separately defined object, remove its name first.

5.23 Conditionals and Loops

`groff` has `if` and `while` control structures like other languages. However, the syntax for grouping multiple input lines in the branches or bodies of these structures is unusual.

5.23.1 Operators in Conditionals

The `if`, `ie`, and `while` requests test the truth values of numeric expressions. They also support several additional Boolean operators. The members of this expanded class are termed *conditional expressions*; their truth values are as shown below.

`c` *chr* — True if a character *chr* is available; *chr* is an ordinary, special or indexed character, whether defined by a font description file or a request.

`d` *nam* — True if a string, macro, diversion, or request named *nam* is defined.

`e` — True if the current page is even-numbered.

[225] See Section 5.24 [Writing Macros], page 182.

F *fnt* True if a font *fnt* is available; *fnt* can be an abstract style or a font name. GNU `troff` handles *fnt* as if it were an argument to the `ft` request (that is, it combines the default family with an abstract style and applies any applicable font translation), but *fnt* cannot be a mounting position, and it does not become mounted.

m *col* True if a color *col* is defined.

n True if the document is being processed in `nroff` mode.

o True if the current page is odd-numbered.

r *reg* True if a register *reg* is defined.

S *sty* True if an abstract style *sty* is registered. Font translation applies.

t True if the document is being processed in `troff` mode.

v Always false. This condition exists for compatibility with certain other `troff` implementations.[226]

If the first argument to an `if`, `ie`, or `while` request begins with a non-alphanumeric character apart from `!` (see below) and is not a numeric expression, the formatter performs an *output comparison test*.[227]

'*xxx*'*yyy*'

This *output comparison expression* interpolates a true value if formatting the comparands *xxx* and *yyy* produces the same output commands. Other delimiters can be used in place of the neutral apostrophes; recall Section 5.6.5 [Delimiters], page 96. `troff` formats *xxx* and *yyy* in separate scratch buffers; after comparison, it discards the resulting data.

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

The resulting glyph properties, including font family, style, size, and slant, must match, but not necessarily the requests and/or escape sequences used to obtain them. In the previous example, '`|`' and '`\fR|\fP`' result in '`|`' glyphs in the same typefaces at the same positions, so the comparands are equal. If '`.ft I`' had been added before the '`.ie`', they would differ: the first '`|`' would produce an italic '`|`', not a roman one. Motions must match in

[226] We refer to `vtroff`, which converted the C/A/T command stream produced by early-vintage AT&T `troff` to input suitable for Versatec and Benson-Varian plotters.

[227] Strictly, letters not otherwise recognized *are* treated as output comparison delimiters. A portable document avoids using letters not in the list above; for example, Plan 9 `troff` uses '`h`' to test a mode it calls `htmlroff`, and GNU `troff` may provide additional operators in the future.

orientation and magnitude to within the applicable horizontal and vertical motion quanta of the device, after rounding. '`.if "\u\d"\v'0'"`' is false even though both comparands result in zero net motion, because motions are not interpreted or optimized but sent as-is to the output.[228] On the other hand, '`.if "\d"\v'0.5m'"`' is true, because `\d` is defined as a downward motion of one-half em.[229]

Surround the comparands with `\?` to avoid formatting them; this causes them to be compared character by character, as with string comparisons in other programming languages.

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

Since GNU `troff` reads comparands protected with `\?` in copy mode,[230] they need not even be syntactically valid. The escape character is still lexically recognized, however, and consumes the next character.

```
.ds a \[
.ds b \[
.if '\?\*a\?'\?\*b\?' a and b equivalent
.if '\?\\?'\?\\?' backslashes equivalent
.if '\?\P\?'\?P\?' backslash-P and P equivalent
    ⇒ a and b equivalent
```

The above operators can't be combined with most others, but a leading '`!`', not followed immediately by spaces or tabs, complements an expression.

```
.nr x 1
.ie !r x register x is not defined
.el     register x is defined
    ⇒ register x is defined
```

Spaces and tabs are optional immediately after the '`c`', '`d`', '`F`', '`m`', '`r`', and '`S`' operators, but right after '`!`', they end the predicate and the conditional evaluates true.[231]

```
.nr x 1
.ie ! r x register x is not defined
.el      register x is defined
    ⇒ r x register x is not defined
```

The unexpected '`r x`' in the output is a clue that our conditional was not interpreted as we planned, but matters may not always be so obvious.

---

[228] Because formatting of the comparands takes place in a dummy environment, vertical motions within them cannot spring traps. See Section 5.29 [Traps], page 206.

[229] All of this is to say that the lists of nodes created by formatting *xxx* and *yyy* must be identical. See Section 5.37 [GNU troff Internals], page 237.

[230] See Section 5.24.2 [Copy Mode], page 188.

[231] This bizarre behavior maintains compatibility with AT&T `troff`.

Conditional operators do not create `roff` language objects as interpolations with `\n` and `\*` escape sequences do.

5.23.2 if-then

`.if` *cond-expr input* [Request]

Evaluate the conditional expression *cond-expr*, and if it evaluates true (or to a positive value), interpret the remainder of the line *input* as if it were an input line. Recall from Section 5.6.2 [Invoking Requests], page 90, that any quantity of spaces between arguments to requests serves only to separate them; leading spaces in *input* are thus not seen. *input* effectively *cannot* be omitted; if *cond-expr* is true and *input* is empty, the formatter interprets the newline at the end of the control line as a blank input line (and therefore a blank text line).

```
super\c
tanker
.nr force-word-break 1
super\c
.if ((\n[force-word-break] = 1) & \n[.int])
tanker
    ⇒ supertanker super tanker
```

`.nop` [*input*] [Request]

Interpret *input* as if it were an input line. `nop` resembles '`.if 1`'; it puts a break on the output if *input* is omitted. Unlike `if`, it cannot govern conditional blocks. Its application is to maintain consistent indentation within macro definitions even when formatting output.

```
.als real-MAC MAC
.de wrapped-MAC
.  tm MAC: called with arguments \\$@
.  nop \\*[real-MAC]\\
..
.als MAC wrapped-MAC
\# Later...
.als MAC real-MAC
```

In the above, we've used aliasing, `nop`, and the interpolation of a macro as a string to interpose a wrapper around the macro '`MAC`' (perhaps to debug it).

5.23.3 if-else

`.ie` *cond-expr input* [Request]
`.el` *input* [Request]

Use the `ie` and `el` requests to write an if-then-else. The first request is the "if" part and the latter is the "else" part. Unusually among programming

languages, any number of non-conditional requests may be interposed between the ie branch and the el branch.

```
.nr a 0
.ie \na a is non-zero.
.nr a +1
.el a was not positive but is now \na.
    ⇒ a was not positive but is now 1.
```

Another way in which el is an ordinary request is that it does not lexically "bind" more tightly to its ie counterpart than it does to any other request. This fact can surprise C programmers.

```
.nr a 1
.nr z 0
.ie \nz \
.  ie \na a is true
.  el     a is false
.el z is false
    ⇒ a is false
```

To conveniently nest conditionals, keep reading.

5.23.4 Conditional Blocks

It is frequently desirable for a control structure to govern more than one request, macro call, text line, or combination of the foregoing. The opening and closing brace escape sequences \{ and \} define such groups. These *conditional blocks* can furthermore be nested.

Brace escape sequences outside of control structures have no meaning (but can be used as delimiters) and produce no output.[232]

\{ [Escape sequence]
\} [Escape sequence]

\{ begins a conditional block; it must appear (after optional spaces and tabs) immediately subsequent to the conditional expression of an if, ie, or while request,[233] or as the argument to an el request.

\} ends a conditional block and should appear on an input line with other occurrences of itself as necessary to match \{ sequences. It can be preceded by a control character, spaces, and tabs. Input after any quantity of opening brace escape sequences \} on the same line is processed only if all of the preceding conditions to which they correspond are true. Further, a closing brace escape sequence \} closing the body of a while request must be the last such escape sequence on an input line.

[232] When its parser can tell that a brace escape sequence is useless, GNU troff emits a warning in category 'style'. See Section 5.38.1 [Warnings], page 245.

[233] See Section 5.23.5 [while], page 181.

Caution: Input lines using `\{` often end with `\RET`, especially in macros that consist primarily of control lines. Forgetting to use `\RET` on an input line after `\{` is a common source of error.

We might write the following in a page header macro. If we delete `\RET`, the header will carry an unwanted extra empty line (except on page 1).

```
.if (\\n[%] != 1) \{\
.  ie ((\\n[%] % 2) = 0) .tl \\*[even-numbered-page-title]
.  el                    .tl \\*[odd-numbered-page-title]
.\}
```

Let us take a closer look at how conditional blocks nest.

```
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
.ie 1 \{\
.  ds a abc\}def
a="\*a"
.el ghi
    ⇒ a="abcdef"
```

The above behavior may challenge the intuition; it was implemented to retain compatibility with AT&T `troff`. A right-brace escape sequence governs flow of control this way: if a branch is *not* taken, then the closing brace that closes a conditional block causes the formatter to ignore the entire input line on which it appears for all purposes except matching braces.

For clarity, it is idiomatic to end input lines with `\{` (followed by `\RET` if appropriate), and to precede `\}` on an input line with nothing more than a control character, spaces, tabs, and other instances of itself.

We can use ie, el, and conditional blocks to simulate the multi-way "switch" or "case" control structures of other languages. The following example is adapted from the groff man package. Indentation is used to clarify the logic.

```
.\" Simulate switch/case in roff.
.      ie '\\$2'1' .ds title General Commands\"
.el \{.ie '\\$2'2' .ds title System Calls\"
.el \{.ie '\\$2'3' .ds title Library Functions\"
.el \{.ie '\\$2'4' .ds title Kernel Interfaces\"
.el \{.ie '\\$2'5' .ds title File Formats\"
.el \{.ie '\\$2'6' .ds title Games\"
.el \{.ie '\\$2'7' .ds title Miscellaneous Information\"
.el \{.ie '\\$2'8' .ds title System Management\"
.el \{.ie '\\$2'9' .ds title Kernel Development\"
.el                .ds title \" empty
.\}\}\}\}\}\}\}\}\}
```

5.23.5 while

GNU troff provides a looping construct: the while request. Its syntax matches the if request.

.while *cond-expr input* [Request]

Evaluate the conditional expression *cond-expr*, and repeatedly execute *input* unless and until *cond-expr* evaluates false. *input*, which is often a conditional block, is referred to as the while request's *body*.

```
.nr a 0 1
.while (\na < 9) \{\
\n+a,
.\}
\n+a
    ⇒ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
```

GNU troff treats the body of a while request similarly to that of a de request (albeit one not read in copy mode[234]), but stores it under an internal name and deletes it when the loop finishes. The operation of a macro containing a while request can slow significantly if its body is large. Each time GNU troff interpolates the macro, it parses and stores the while body again.

```
.de xxx
.  nr num 10
.  while (\\n[num] > 0) \{\
.    \" many lines of code
.    nr num -1
.  \}
..
```

[234] See Section 5.24.2 [Copy Mode], page 188.

Often, a better solution—and a more portable one, since AT&T `troff` lacked the `while` request—is to instead write a recursive macro, which is parsed only once.[235]

```
.de yy
.  if (\\n(nm > 0) \{\
.    \" many lines of code
.    nr nm -1
.    yy
.  \}
..
.
.de xx
.  nr nm 10
.  yy
..
```

To prevent infinite loops, GNU `troff` limits the default number of available recursion levels to 1,000 or somewhat less.[236] You can disable this protective measure, or alter the limit, by setting the `slimit` register. See Section 5.38 [Debugging], page 240.

As noted above, if a `while` body begins with a conditional block, its closing brace must end an input line.

```
.if 1 \{\
.  nr a 0 1
.  while (\n[a] < 10) \{\
.    nop \n+[a]
.\}\}
```

error `unbalanced brace escape sequences`

`.break` [Request]

Exit a `while` loop. Do not confuse this request with a typographical break or the `br` request.

`.continue` [Request]

Skip the remainder of a `while` loop's body, immediately retesting its conditional expression.

5.24 Writing Macros

A *macro* is a stored collection of text and control lines that can be interpolated multiple times. Use macros to define common operations. Macros are called in the same way that requests are invoked. While requests exist for the purpose of creating macros, simply calling an undefined macro, or interpolating it as a string, causes it to be defined as empty. A macro's name is an identifier; recall Section 5.5 [Identifiers], page 86.

[235] unless you redefine it

[236] "somewhat less" because things other than macro calls can be on the input stack

`.de` *name* [*end*] [Request]

Define a macro *name*, replacing the definition of any existing request, macro, string, or diversion called *name*. In GNU `troff`, if *name* already exists as an alias, `de` redefines the target of the alias; recall Section 5.22 [Strings], page 170. The formatter enters copy mode,[237] storing subsequent input lines as the macro definition. *end* identifies a macro whose call syntax at the start of a control line ends the definition of *name*; the formatter then calls *end* normally. Omitting the *end* argument causes a subsequent control line '`..`' (two dots) to end the definition. A macro definition must end in the same conditional block (if any) in which it began; recall Section 5.23.4 [Conditional Blocks], page 179. Spaces or tabs are permitted after the control character in the line containing this ending token (either '`.`' or '*end*'), but a tab immediately after the token prevents its recognition as the end of a macro definition. The macro *end* can be called with arguments.[238]

Here is a small example macro called '`P`' that causes a break and inserts some vertical space. It could be used to separate paragraphs.

```
.de P
.  br
.  sp .8v
..
```

We can define one macro within another. Attempting to nest '`..`' naïvely will end the outer definition because the inner definition isn't interpreted as such until the outer macro is later interpolated. We can use an end macro instead. Each level of nesting should use a unique end macro.

An end macro need not be defined until it is called. This fact enables a nested macro definition to begin inside one macro and end inside another. Consider the following example.[239]

---

[237] See Section 5.24.2 [Copy Mode], page 188.

[238] While it is possible to define and call a macro '`.`', you can't use it as an end macro: during a macro definition, '`..`' is never handled as calling '`.`', even if '`.de` *name* `.`' explicitly precedes it.

[239] Its structure is adapted from, and isomorphic to, part of a solution by Tadziu Hoffman to the problem of reflowing text multiple times to find an optimal configuration for it. `https://lists.gnu.org/archive/html/groff/2008-12/msg00006.html`

```
.de m1
.  de m2 m3
you
..
.de m3
Hello,
Joe.
..
.de m4
do
..
.m1
know?
.  m3
What
.m4
.m2
    ⇒ Hello, Joe.  What do you know?
```

A nested macro definition *can* be terminated with '`..`' and nested macros *can* reuse end macros, but these control lines must be escaped multiple times for each level of nesting. The necessity of this escaping and the utility of nested macro definitions will become clearer when we employ macro parameters and consider the behavior of copy mode in detail.

`de` defines a macro that inherits the compatibility mode enablement status of its context (see Section 5.39 [Implementation Differences], page 248). Often it is desirable to make a macro that uses `groff` features callable from contexts where compatibility mode is on; for instance, when writing extensions to a historical macro package. To achieve this, compatibility mode needs to be switched off while such a macro is interpreted—without disturbing that state when it is finished.

`.de1` *name* [*end*] [Request]

The `de1` request defines a macro to be interpreted with compatibility mode disabled. When *name* is called, compatibility mode enablement status is saved; it is restored when the call completes. Observe the extra backlash before the interpolation of register '`xxx`'; we'll explore this subject in Section 5.24.2 [Copy Mode], page 188.

```
.nr xxx 12345
.de aa
The value of xxx is \\n[xxx].
.  br
..
.de1 bb
The value of xxx is \\n[xxx].
..
.cp 1
.aa
    error  warning: register '[' not defined
    ⇒ The value of xxx is 0xxx].
.bb
    ⇒ The value of xxx is 12345.
```

`.dei` *name* [*end*] [Request]
`.dei1` *name* [*end*] [Request]

The `dei` request defines a macro with its name and end macro indirected through strings. That is, it interpolates strings named *name* and *end* before performing the definition.

The following examples are equivalent.

```
.ds xx aa
.ds yy bb
.dei xx yy
.de aa bb
```

The `dei1` request bears the same relationship to `dei` as `de1` does to `de`; it temporarily turns compatibility mode off when *name* is called.

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

`am` appends subsequent input lines to macro *name*, extending its definition, and otherwise working as `de` does.

To make the previously defined '`P`' macro set indented instead of block paragraphs, add the necessary code to the existing macro.

```
.am P
.ti +5n
..
```

The other requests are analogous to their '`de`' counterparts. The `am1` request turns off compatibility mode during interpretation of the appendment. The `ami` request appends indirectly, meaning that strings *name* and *end* are interpolated with the resulting names used before appending. The `ami1` request is similar to `ami`, disabling compatibility mode during interpretation of the appended lines.

Using `trace.tmac`, you can trace calls to `de`, `de1`, `am`, and `am1`. You can also use the `backtrace` request at any point desired to troubleshoot tricky spots (see Section 5.38 [Debugging], page 240).

See Section 5.22 [Strings], page 170, for the `als`, `rm`, and `rn` requests to create an alias of, remove, and rename a macro, respectively.

Macro identifiers share their name space with requests, strings, and diversions; see Section 5.5 [Identifiers], page 86. The `am`, `as`, `da`, `de`, `di`, and `ds` requests (together with their variants) create a new object only if the name of the macro, diversion, or string is currently undefined or if it is defined as a request; normally, they modify the value of an existing object. See [the description of the `als` request], page 174, for pitfalls when redefining a macro that is aliased.

`.return` [*input*] [Request]

Stop interpreting an interpolated macro, skipping to the end of its definition. Do not confuse `return` with `rt`. If called with an argument *input*, GNU `troff` performs the skip twice—once within the macro being interpreted and once in an enclosing macro, permitting a macro to wrap the request.[240]

5.24.1 Parameters

Macro calls and string interpolations optionally accept a list of arguments; recall Section 5.6.3 [Calling Macros], page 92. At the time such an interpolation takes place, these *parameters* can be examined using a register and a variety of escape sequences starting with '`\$`'. All such escape sequences are interpreted even in copy mode, a fact we shall motivate and explain below (see Section 5.24.2 [Copy Mode], page 188).

`\n[.$]` [Register]

The read-only register `.$` makes the count of a macro or interpolation's parameters available to its definition. The `shift` request dequeues parameters, and thus updates `.$`'s value.

Any individual parameter can be accessed by its position in the list of arguments to the macro call, numbered from left to right starting at 1, with one of the following escape sequences.

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

Interpolate the *n*th, *nn*th, or *nnn*th parameter. The first form expects only a single digit ($1 \le n \le 9$)), the second two digits ($01 \le nn \le 99$)), and the third any positive integer *nnn*. Macros and strings accept an unlimited number of parameters. The formatter interprets `\$` even in copy mode.[241]

[240] as `trace.tmac` does

[241] See Section 5.24.2 [Copy Mode], page 188.

`.shift` [*n*] [Request]

Shift macro or string parameters *n* places (by 1 if *n* omitted): argument *i* becomes argument *i*-*n*; arguments 1 to *n* become unavailable. The register `.$` adjusts its value accordingly. Shifting by a non-positive amount, or outside of a macro or string definition, performs no operation.

In practice, parameter interpolations are usually seen prefixed with an extra escape character. This is because the `\$` family of escape sequences is interpreted even in copy mode.[242]

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

In some cases it is convenient to interpolate all of the parameters at once (to pass them to a request, for instance). The `\$*` escape catenates the parameters, separating them with spaces. `\$@` is similar, surrounding each parameter with double quotes and separating them with spaces. If not in compatibility mode, the interpolation depth of double quotes is preserved (see Section 5.6.3 [Calling Macros], page 92). `\$^` interpolates all parameters as if they were arguments to the `ds` request.

```
.de foo
. tm $1='\\$1'
. tm $2='\\$2'
. tm $*='\\$*'
. tm $@='\\$@'
. tm $^='\\$^'
..
.foo " This is a "test"
     error  $1=' This is a '
     error  $2='test"'
     error  $*=' This is a  test"'
     error  $@='" This is a " "test""'
     error  $^='" This is a "test"'
```

`\$*` is useful when writing a macro that doesn't need to distinguish its arguments, or even to not interpret them; examples include macros that produce diagnostic messages by wrapping the `tm` or `ab` requests. Use `\$@` when writing a macro that may need to shift its parameters and/or wrap a macro or request that finds the count significant. If in doubt, prefer `\$@` to `\$*`. An application of `\$^` is seen in `trace.tmac`, which redefines some requests and macros for debugging purposes.

---

[242] If they were not, parameter interpolations would be similar to command-line parameters—fixed for the entire duration of a `roff` program's run. The advantage of interpolating `\$` escape sequences even in copy mode is that they can interpolate different contents from one call to the next, like function parameters in a procedural language. The additional escape character is the price of this power.

`\$0` [Escape sequence]

Interpolate the name by which the macro being interpreted was called. The `als` request can cause a macro to have more than one name. Applying string interpolation to a macro does not change this name.[243]

```
.de foo
.  tm \\$0
..
.als bar foo
.
.de aaa
.  foo
..
.de bbb
.  bar
..
.de ccc
\\*[foo]\\
..
.de ddd
\\*[bar]\\
..
.
.aaa
     error  foo
.bbb
     error  bar
.ccc
     error  ccc
.ddd
     error  ddd
```

5.24.2 Copy Mode

The formatter processes certain requests in *copy mode*: it copies ordinary, special, and indexed characters as-is; interpolates the escape sequences `\n`, `\g`, `\$`, `\*`, `\V`, and `\?` normally; discards comments `\"` and `\#`; interpolates `\a`, `\e`, and `\t`, as the current leader, escape, or tab character, respectively; represents `\RET`, `\&`, `\_`, `\|`, `\^`, `\`, `\'`, `\-`, `\!`, `\c`, `\%`, `\SPC`, `\E`, `\)`, `\~`, and `\:` in an encoded form, and copies other escape sequences as-is. (When defining a macro, GNU `troff` also encodes `\{` and `\}` so that they can participate in control flow decisions; recall Section 5.23.4 [Conditional Blocks], page 179.) The term "copy mode" reflects its most visible application in requests that populate macros and strings, but other requests also use it when interpreting arguments that can't meaningfully represent typesetting operations. For example, a font selection escape sequence has no meaning

[243] See Section 5.31 [Punning Names], page 222.

in a hyphenation pattern file name (hpf) or a diagnostic message written to the terminal (tm). Any other use of the escape character in copy mode serves to *quote* the subsequent character, delaying its interpretation until the material processed in copy mode is subsequently interpolated.

The complement of copy mode—a roff formatter's behavior when not defining or appending to a macro, string, or diversion—where all macros are interpolated, requests invoked, and valid escape sequences processed immediately upon recognition, can be termed *interpretation mode*.

\\ [Escape sequence]

Quoting the escape character enables you to control whether the formatter interprets a given \n, \g, \$, *, \V, or \? escape sequence at the time the macro containing it is defined, or later when the macro is called.[244]

```
.nr x 20
.de y
.nr x 10
\&\nx
\&\\nx
..
.y
    ⇒ 20 10
```

\. [Escape sequence]

\. quotes the control character. It is used to permit nested macro definitions to end without a named macro call to conclude them. Without a syntax for quoting the control character, this would not be possible.

```
.de m1
foo
.  de m2
bar
\\..
..
.m1
.m2
    ⇒ foo bar
```

The first backslash is consumed while the macro is read, and the second is interpreted when macro m1 is called.

Outside of copy mode, roff documents should not use the \\ or \. character sequences; they serve only to obfuscate the input. Use \e to represent the escape character, \[rs] to obtain a backslash glyph, and \& before '.' and '\'' where the formatter expects them as control characters if you mean to use them literally; recall Section 5.1.7 [Requests and Macros], page 71.

[244] Compare this to the \def and \edef commands in TEX.

Macro definitions can be nested to arbitrary depth. The mechanics of parsing the escape character have significant consequences for this practice.

```
.de M1
\\$1
.  de M2
\\\\$1
.    de M3
\\\\\\\\$1
\\\\..
.    M3 hand.
\\..
.  M2 of
..
This understeer is getting
.M1 out
    ⇒ This understeer is getting out of hand.
```

As seen above, the formatter interprets each escape character in multiple contexts; once, when populating the macro or string, where the first '\' serves its quotation function—thus only one '\' is stored in the definition. (Verify this fact with the `pm` request.) The formatter interprets the second '\' as an escape character (assuming the escape character hasn't been changed in the meantime) each time it interpolates the macro or string definition. This fact leads to exponential growth in the quantity of escape characters required to quote and thereby delay interpolation of `\n`, `\g`, `\$`, `\*`, `\V`, and `\?` at each nesting level, which can be daunting. GNU `troff` offers a solution.

`\E` [Escape sequence]

`\E` encodes an escape character that is not interpreted in copy mode. Its use can ease the writing of nested macro definitions.

```
.de M1
.  nop \E$1
.  de M2
.    nop \E$1
.    de M3
.      nop \E$1
\\\\..
.    M3 better.
\\..
.  M2 bit
..
This vehicle handles
.M1 a
    ⇒ This vehicle handles a bit better.
```

Because '`\.`' is a quoted control character, we can't use `\E` to keep '`..`' from ending a macro definition prematurely. If the multiplicity of backslashes complicates maintenance, use end macros.

\E is also convenient to define strings containing escape sequences that need to work when used in copy mode (for example, as macro arguments), or that are interpolated at varying macro nesting depths. We might define strings to begin and end superscripting as follows.[245]

```
.ds { \v'-.9m\s'\En[.s]*7u/10u'+.7m'
.ds } \v'-.7m\s0+.9m'
```

When the ec request is used to redefine the escape character, \E also makes it easier to distinguish the semantics of an escape character from the other meaning(s) its character might have. Consider the use of an unusual escape character, '-'.

```
.nr a 1
.ec -
.de xx
--na
..
.xx
    ⇒ -na
```

This result may surprise you; some people expect '1' to be output since register 'a' has clearly been defined with that value. What has happened? The robotic replacement of '\' with '-' has led us astray. You might recognize the sequence '--' more readily with the default escape character as '\-', the special character escape sequence for the minus sign glyph.

```
.nr a 1
.ec -
.de xx
-Ena
..
.xx
    ⇒ 1
```

5.25 Page Motions

See Section 5.11 [Manipulating Spacing], page 122, for a discussion of the most commonly used request for vertical motion, sp.

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

You can *mark* a location on a page for subsequent *return*. mk takes an argument, a register name in which to store the current page location. If given no argument, it stores the location in an internal register. This location can be used later by the rt or the sp requests (or the \v escape sequence).

The rt request returns *upward* to the location marked with the last mk request. If used with an argument, it returns to a vertical position *dist*

[245] These are lightly adapted from the groff implementation of the ms macros.

from the top of the page (no previous call to `mk` is necessary in this case). The default scaling unit is '`v`'.

If a page break occurs between a `mk` request and its matching `rt` request, the `rt` request is silently ignored.

A simple implementation of a macro to set text in two columns follows. This example also defines a macro to be called when a trap is sprung;[246] this trap macro performs the motion to the next column.

```
.nr column-length 1.5i
.nr column-gap 4m
.nr bottom-margin 1m
.
.de 2c
.  br
.  mk
.  ll \\n[column-length]u
.  wh -\\n[bottom-margin]u 2c-trap
.  nr right-side 0
..
.
.de 2c-trap
.  ie \\n[right-side] \{\
.    nr right-side 0
.    po -(\\n[column-length]u + \\n[column-gap]u)
.    \" remove trap
.    wh -\\n[bottom-margin]u
.  \}
.  el \{\
.    \" switch to right side
.    nr right-side 1
.    po +(\\n[column-length]u + \\n[column-gap]u)
.    rt
.  \}
..
```

Now let us apply our two-column macro.

[246] See Section 5.29.1.1 [Page Location Traps], page 206.

```
.pl 1.5i
.ll 4i
This is a small test that shows how the
rt request works in combination with mk.

.2c
Starting here, text is typeset in two columns.
Note that this implementation isn't robust
and thus not suited for a real two-column
macro.
    ⇒ This is a small test that shows how the
    ⇒ rt request works in combination with mk.
    ⇒
    ⇒ Starting  here,    isn't    robust
    ⇒ text is typeset    and   thus  not
    ⇒ in two columns.    suited  for   a
    ⇒ Note that  this    real two-column
    ⇒ implementation     macro.
```

Several escape sequences enable fine control of movement about the page.

\v'*expr*' [Escape sequence]

Vertically move the drawing position. *expr* indicates the magnitude of motion: positive is downward and and negative upward. The default scaling unit is 'v'. The motion is relative to the current drawing position unless *expr* begins with the boundary-relative measurement operator '|'; recall Section 5.4 [Numeric Expressions], page 82.

Text processing continues at the new drawing position; usually, vertical motions should be in balanced pairs to avoid a confusing page layout.

\v does not spring a vertical position trap. This can be useful; for example, consider a page bottom trap macro that prints a mark in the margin to indicate continuation of a footnote.[247]

A few escape sequences that produce vertical motion are unusual. They are thought to originate early in AT&T nroff history to achieve super- and subscripting by half-line motions on line printers and teletypewriters before the phototypesetter made more precise positioning available. They are reckoned in ems—not vees—to maintain continuity with their original purpose of moving relative to the size of the type rather than the distance between text baselines (vees).[248]

[247] See Section 5.29 [Traps], page 206.

[248] At the grops defaults of 10-point type on 12-point vertical spacing, the difference between half a vee and half an em can be subtle: large spacings, as configured by '.vs .5i' for example, make it obvious.

`\r` [Escape sequence]
`\u` [Escape sequence]
`\d` [Escape sequence]

Move upward 1 m, upward .5 m, and downward .5 m, respectively.

Let us see these escape sequences in use.

```
Obtain 100 cm\u3\d of \ka\d\092\h'|\nau'\r233\dU.
```

In the foregoing we have paired `\u` and `\d` to typeset a superscript, and later a full em negative ("reverse") motion to place a superscript above a subscript. A numeral-width horizontal motion escape sequence aligns the proton and nucleon numbers, while `\k` marks a horizontal position to which `\h` returns so that we could stack them. (We shall discuss these horizontal motion escape sequences presently.) In serious applications, we often want to alter the type size of the -scripts and to fine-tune the vertical motions, as the `groff ms` package does with its super- and subscripting string definitions.

`\h'`*expr*`'` [Escape sequence]

Horizontally move the drawing position. *expr* indicates the magnitude of motion: positive is rightward and negative leftward. The default scaling unit is '`m`'. The motion is relative to the current drawing position unless *expr* begins with the boundary-relative measurement operator '`|`'. See Section 5.4 [Numeric Expressions], page 82.

The following string definition sets the TEX logo. Recall Section 5.22 [Strings], page 170, regarding the trailing '`\"`'.

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

An input backspace becomes a negative horizontal motion of one word space; recall Section 5.9 [Manipulating Filling and Adjustment], page 106. This feature persists for backward compatibility with early formatters that predate `nroff` and even Unix itself, and which used it to facilitate user-directed overstriking for character composition, boldfacing, and underlining. GNU `troff` has explicit features to support each of these; use them instead.

Several escape sequences support special cases of horizontal motion.

`\SPC` [Escape sequence]

Move right one word space. (The input is a backslash followed by a space.) This escape sequence can be thought of as a non-adjustable, unbreakable space. Usually you want `\~` instead; recall Section 5.9 [Manipulating Filling and Adjustment], page 106.

`\|` [Escape sequence]

Move one-sixth em to the right on typesetting output devices. If a glyph named '`\|`' is defined in the current font, its width is used instead, even on terminal output devices.

\^ [Escape sequence]

Move one-twelfth em to the right on typesetting output devices. If a glyph named '\^' is defined in the current font, its width is used instead, even on terminal output devices.

\0 [Escape sequence]

Move right by the width of a numeral in the current font.

Horizontal motions are not discarded at the end of an output line as word spaces are; recall Section 5.1.4 [Breaking], page 70.

\w'*input*' [Escape sequence]
\n[st] [Register]
\n[sb] [Register]
\n[rst] [Register]
\n[rsb] [Register]
\n[ct] [Register]
\n[ssc] [Register]
\n[skw] [Register]

Interpolate the width of *input*, as interpreted, in basic units. This escape sequence allows several properties of formatted output to be measured without writing it out.

```
The length of the string 'abc' is \w'abc'u.
    ⇒ The length of the string 'abc' is 72u.
```

The formatter processes *input* in a dummy environment: this means that font and type size changes, for example, may occur within it without affecting subsequent output.

After each use, \w sets several registers.

st
sb — The maximum vertical displacements of the text baseline above and below, respectively. The sign convention is opposite that of relative vertical motions; that is, depth below the (original) baseline is negative. These registers are incorrectly documented in the AT&T troff manual as "the highest and lowest extent of [the argument to \w] relative to the baseline".

rst
rsb — Like st and sb, but taking account of the heights and depths of glyphs. In other words, these registers store the highest and lowest vertical positions attained by *input*, doing what AT&T troff documented st and sb as doing.

ct — Characterizes the geometry of glyphs occurring in *input*.

0	only short glyphs, no descenders or tall glyphs
1	at least one descender
2	at least one tall glyph

3 at least one each of a descender and a tall glyph

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

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

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

Store the horizontal drawing position, relative to that corresponding to the start of the input line (ignoring page offset and indentation), in a register with the name *position* (one-character name *p*, two-character name *ps*). Use this, for example, to later move to the beginning of a word for highlighting or other decoration.

`\n[hp]` [Register]

The horizontal position relative to that at the start of the input line.

`\n[.k]` [Register]

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

`\o'`*abc...*`'` [Escape sequence]

Overstrike the glyphs of characters *a*, *b*, *c*, ...; the glyphs are centered, written, and the drawing position advanced by the widest of the glyphs.

`\z`*c* [Escape sequence]

Format the character *c* with zero width; that is, without advancing the drawing position. Use `\z` to overstrike glyphs aligned to their left edges, in contrast to `\o`'s centering.

`\Z'`*input* [Escape sequence]

Save the drawing position, format *input*, then restore the saved position. GNU `troff` disallows tabs and leaders in *input*.

We might implement a strike-through macro thus.

```
.de strikeout
.nr width \w'\\$1'
\Z@\v'-.25m'\l'\\n[width]u'@\\$1
..
.
This is
.strikeout "a test"
an actual emergency!
```

5.26 Output Line Annotation

After an output line is broken (and adjusted, if applicable), it can be annotated in the margins. You can indicate line numbers on the left, and apply a margin character on the right.

`.nm` [*start* [*increment* [*space* [*indentation*]]]] [Request]
`\n[ln]` [Register]
`\n[.nm]` [Register]

Begin (or, with no arguments, cease) numbering output lines. *start* assigns the number of the *next* output line. Only line numbers divisible by *increment* (default: '`1`') bear numeric marks. The formatter reckons the third and fourth arguments in numeral widths (`\0`): *space* configures the horizontal spacing between the number and the text (default: '`1`'). Any given *indentation* applies to the numbers (default: '`0`'). *start* must be non-negative and *increment* positive. Line numbering parameters are associated with the environment.[249]

The formatter aligns the number to the right in a space of three numeral widths plus *indentation*, then catenates *space* and the output line. The line length is *not* reduced. Depending on the value of the page offset (recall Section 5.15 [Line Layout], page 131) numbers wider than the allocated space protrude into the left margin, or shift the output line to the right.

Line numbering parameters corresponding to missing arguments are not altered. After numbering is disabled, '`.nm +0`' resumes it using the previously active parameters.

While numbering is enabled, the output line number register `ln` is updated as each line is output, even if no line number is formatted with it because it is being skipped (it is not a multiple of *increment*) or because numbering is suppressed (see the `nn` request below).

The `.nm` read-only register tracks the enablement status of numbering. Temporary suspension of numbering with the `nn` request does *not* alter its value.

A simple example follows.

```
.ll 45n
.nm 1
Searches for matching strings of elemental units are
conducted in a parallel fashion such that the search may
be conducted concurrently on different substrings.
    ⇒   1 Searches  for  matching  strings of elemental
    ⇒   2 units are conducted  in  a  parallel  fashion
    ⇒   3 such that the search may be conducted concur-
    ⇒   4 rently on different substrings.
```

[249] See Section 5.32 [Environments], page 224.

The following example illustrates all of the nm request's features.

```
.po 5n
.ll 44n
Programming,
when stripped of all its circumstantial irrelevancies,
.nm 999 1 1 -4
boils down to no more and no less than
.nm +0 3
very effective thinking so as to avoid unmastered
.nn 2
complexity,
to very vigorous separation of your many
different concerns.
.br
\(em Edsger Dijkstra
.sp
.nm 1 1 1
This guy's arrogance takes your breath away.
.br
\(em John Backus
```

```
    ⇒      Programming,  when  stripped of all its cir-
    ⇒  999 cumstantial irrelevancies, boils down to  no
    ⇒      more  and no less than very effective think-
    ⇒      ing so as to avoid unmastered complexity, to
    ⇒      very vigorous separation of your  many  dif-
    ⇒      ferent concerns.
    ⇒ 1002 -- Edsger Dijkstra
    ⇒
    ⇒    1 This guy's arrogance takes your breath away.
    ⇒    2 -- John Backus
```

.nn [*skip*] [Request]
\n[.nn] [Register]

Suppress numbering of the next *skip* output lines counted by the nm request. If *skip* is '0', cancel suppression. The default is 1. nn can be invoked when line numbering is not active; suppression of numbering takes effect for *skip* lines once nm enables it. This count is associated with the environment.[250]

The .nn read-only register stores the count of lines remaining in the environment for which numbering is suppressed while output line numbering is enabled.

[250] See Section 5.32 [Environments], page 224.

To test whether the current output line will be numbered, you must check both the .nm and .nn registers.

```
.de is-numbered
.  nop This line
.  ie (\\n[.nm] & (1-\\n[.nn])) IS
.  el                           ISN'T
.  nop numbered.
.  br
..
Test line numbering.
.is-numbered
.nm 1
.nn 1
.is-numbered
.is-numbered
.nm
.is-numbered
  ⇒ Test line numbering.  This line ISN'T numbered.
  ⇒ This line ISN'T numbered.
  ⇒   1 This line IS numbered.
  ⇒ This line ISN'T numbered.
```

.mc [*margin-character* [*distance*]] [Request]

Begin (or, with no arguments, cease) writing a *margin-character* to the right of each output line. The *distance* argument separates *margin-character* from the right margin. If absent, the most recent value is used; the default is 10 points. If an output line exceeds the line length, the margin character is appended to it. No margin character is written on lines produced by the tl request. The margin character is associated with the environment.[251]

The margin character is a property of the output line. Only one margin character is in effect at one time; the most recent mc call determines its value. If the margin character is disabled before an output line breaks, none is output (but see below).

[251] See Section 5.32 [Environments], page 224.

```
.ll 5i
.nf
.mc \[br]
This paragraph is marked with a margin character.
.sp
As seen above, vertical space isn't thus marked.
\&
An output line that is present, but empty, is.
    ⇒ This paragraph is marked with a margin character.  |
    ⇒                                                    |
    ⇒ As seen above, vertical space isn't thus marked.   |
    ⇒                                                    |
    ⇒ An output line that is present, but empty, is.     |
```

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.

```
.ll 10n
.nf
.mc |
.mc *
.mc
foo
bar
    ⇒ foo        *
    ⇒ bar
```

The margin character mechanism is commonly used to annotate changes in documents. The `groff` distribution ships a program, `gdiffmk`, to assist with this task.[252]

5.27 Drawing Geometric Objects

A few of the formatter's escape sequences draw lines and other geometric objects. Combined with each other and with page motion commands (see Section 5.25 [Page Motions], page 191), a wide variety of figures is possible. For complex drawings, these operations can be cumbersome: the preprocessors `pic` or `grn` are typically used instead.

The `\l` and `\L` escape sequences draw horizontal and vertical sequences of glyphs, respectively. Even the simplest of output devices supports them. They require an argument specifying the length of the rule (line) to be drawn, optionally followed by a single ordinary or special character with which to draw the rule if the default is not desired. If the character is valid in a numerical expression, put `\&` after *l* to disambiguate the input.

[252] Historically, tools named `nrchbar` and `changebar` were developed for marking changes with margin characters and could be found in archives of the `comp.sources.unix` Usenet group. Some proprietary Unices also offer(ed) a `diffmk` program.

\l'*l*' [Escape sequence]
\l'*lc*' [Escape sequence]

Draw a horizontal line of length *l* from the drawing position. Rightward motion is positive. Afterward, the drawing position is at the right end of the line. The default scaling unit is 'm'. The default glyph is the baseline rule special character, \[ru].

```
\l'4i\&-'
    ⇒ ----------------------------------------
```

Let us see how to draw a box around a word using a macro.

```
.de textbox
\[br]\\$*\[br]\l'|0\[rn]'\l'|0\[ul]'
..
```

The foregoing outputs a box rule (a vertical line), the text argument(s), and another box rule. We employ the boundary-relative measurement operator '|'. Finally, the line-drawing escape sequences draw a radical extender (a form of overline) and an underline from the drawing position to the position corresponding to beginning of the *input* line. The formatter leaves the drawing position at the right-hand box rule even though the line lengths are negative, as noted above.

\L'*l*' [Escape sequence]
\L'*lc*' [Escape sequence]

Draw a vertical line of length *l* from the drawing position. Downward motion is positive. The default scaling unit is 'v'. The default glyph is the box rule, \[br]. As with vertical motion escape sequences, text processing continues where the line ends.

```
$ nroff <<EOF
This is a \L'3v'test.
EOF
    ⇒ This is a
    ⇒           |
    ⇒           |
    ⇒           |test.
```

When writing text, the drawing position is at the text baseline; recall Section 5.2 [Page Geometry], page 79.

The \D escape sequence provides *drawing commands* that direct the output device to render geometrical objects rather than glyphs. Specific devices may support only a subset, or may feature additional ones; consult the man page for the output driver in use. Terminals in particular implement almost none. See Section 6.2.2.3 [Graphics Commands], page 268.

Rendering starts at the drawing position; when finished, the drawing position is left at the rightmost point of the object, even for closed figures, except where noted. GNU troff draws stroked (outlined) objects with the stroke color, and shades filled ones with the fill color. See Section 5.21

[Colors], page 168. Coordinates *h* and *v* are horizontal and vertical motions relative to the drawing position or previous point in the command. The default scaling unit for horizontal measurements (and diameters of circles) is '`m`'; for vertical ones, '`v`'.

Circles, ellipses, and polygons can be drawn filled or stroked. These are independent properties; if you want a filled, stroked figure, you must draw the same figure twice using each drawing command. A filled figure is always smaller than an outlined one because the former is drawn only within its defined area, whereas strokes have a line thickness (set with '`\D't`').

```
\h'1i'\v'1i'\
\# increase line thickness
\Z'\D't 5p''\
\# draw stroked (unfilled) polygon
\Z'\D'p 3 3 -6 0''\
\# draw filled (solid) polygon
\Z'\D'P 3 3 -6 0''
```

`\D'`*command argument* . . .`'` [Escape sequence]

Drawing command escape sequence parameters begin with an ordinary character, *command*, selecting the type of object to be drawn, followed by *arguments* whose meaning is determined by *command*.

`\D'~` *h1 v1* . . . *hn vn*`'`
: Draw a B-spline to each point in sequence, leaving the drawing position at (*hn*, *vn*).

`\D'a` *hc vc h v*`'`
: Draw a circular arc centered at (*hc*, *vc*) counterclockwise from the drawing position to a point (*h*, *v*) relative to the center.[253]

`\D'c` *d*`'`
: Draw a circle of diameter *d* with its leftmost point at the drawing position.

`\D'C` *d*`'`
: As '`\D'C` . . .`'`', but the circle is filled.

`\D'e` *h v*`'`
: Draw an ellipse of width *h* and height *v* with its leftmost point at the drawing position.

`\D'E` *x y*`'`
: As '`\D'e` . . .`'`', but the ellipse is filled.

`\D'l` *dx dy*`'`
: Draw line from the drawing position to (*h*, *v*).

  The following is a macro for drawing a box around a text argument; for simplicity, the box margin is a fixed at 0.2 m.

[253] (*hc*, *vc*) is adjusted to the point nearest the perpendicular bisector of the arc's chord.

```
.de TEXTBOX
.  nr @wd \w'\\$1'
\h'.2m'\
\h'-.2m'\v'(.2m - \\n[rsb]u)'\
\D'l 0 -(\\n[rst]u - \\n[rsb]u + .4m)'\
\D'l (\\n[@wd]u + .4m) 0'\
\D'l 0 (\\n[rst]u - \\n[rsb]u + .4m)'\
\D'l -(\\n[@wd]u + .4m) 0'\
\h'.2m'\v'-(.2m - \\n[rsb]u)'\
\\$1\
\h'.2m'
..
```

The argument is measured with the `\w` escape sequence. Its width is stored in register `@wd`. `\w` also sets the registers `rst` and `rsb`; these contain its maximum vertical extents of the argument. Then, four lines are drawn to form a box, offset by the box margin.

`\D'p` *h1 v1 ... hn vn*`'`

Draw polygon with vertices at the drawing position and each point in sequence. GNU `troff` closes the polygon by drawing a line from (*hn*, *vn*) back to the initial drawing position and leaves the drawing position there.[254]

`\D'P` *dx1 dy1 dx2 dy2 ...*`'`

As '`\D'P ...'`', but the polygon is filled. `groff` does not specify how the output device must fill concave or self-intersecting polygons.

The following macro is like the '`\D'l'`' example, but shades the box. We draw the box before writing the text because colors in GNU `troff` have no transparency; in the opposite order, the filled polygon would occlude the text.

[254] DWB and Heirloom Doctools `troff`s do neither.

```
.de TEXTBOX
.  nr @wd \w'\\$1'
\h'.2m'\
\h'-.2m'\v'(.2m - \\n[rsb]u)'\
\M[lightcyan]\
\D'P 0 -(\\n[rst]u - \\n[rsb]u + .4m) \
     (\\n[@wd]u + .4m) 0 \
     0 (\\n[rst]u - \\n[rsb]u + .4m) \
     -(\\n[@wd]u + .4m) 0'\
\h'.2m'\v'-(.2m - \\n[rsb]u)'\
\M[]\
\\$1\
\h'.2m'
..
```

`\D't` *n*`'` Set the stroke thickness of geometric objects to *n* basic units. A zero *n* selects the minimum supported thickness. A negative *n* selects a thickness proportional to the type size; this is the default.

In a hazy penumbra between text rendering and drawing commands we locate the bracket-building escape sequence, `\b`. It can assemble glyphs that appear large by vertically stacking ordinary ones.

`\b'`*contents*`'` [Escape sequence]

Pile and center a sequence of glyphs vertically on the output line. *Piling* stacks glyphs corresponding to each character in *contents*, read from left to right, and placed from top to bottom. GNU `troff` separates the glyphs vertically by 1 m, and the pile itself is centered 0.5 m above the text baseline. The horizontal drawing position is then advanced by the width of the widest glyph in the pile.

This rather inflexible positioning algorithm doesn't work with the `dvi` output device since its bracket pieces vary in height. Instead, use the `eqn` preprocessor.

Section 5.11 [Manipulating Spacing], page 122, describes how to adjust the vertical spacing of the output line with the `\x` escape sequence.

The application of `\b` that lends its name is construction of brackets, braces, and parentheses when typesetting mathematics. We might construct a large opening (left) brace as follows.

```
\b'\[lt]\[bv]\[lk]\[bv]\[lb]'
```

See *groff_char*(7) for a list of special character identifiers.

5.28 Deferring Output

A few `roff` language elements are generally not used in simple documents, but arise as page layouts become more sophisticated and demanding. *Environments* collect formatting parameters like line length and typeface. A

diversion stores formatted output for later use. A *trap* is a condition on the input or output, tested automatically by the formatter, that is associated with a macro: fulfilling the condition *springs* the trap—calls the macro.

Footnote support often exercises all three of the foregoing features. A simple implementation might work as follows. The author writes a pair of macros: one starts a footnote and the other ends it. The author also sets a trap a small distance above the page bottom, reserving a footnote area. The document calls the first macro where a footnote mark is desired. This macro establishes a diversion so that the footnote text is collected at the place in the body text where its corresponding mark appears. It further creates and selects an environment for the footnote so that it sets using a smaller type size and vertical spacing. Diverting the text formats it, but it does not yet appear in the output. The document calls the footnote end macro, which returns to the previous environment and ends the diversion. Later, after body text nearly fills the page, the trap springs. The macro called by the trap draws a line across the page and emits the stored diversion by calling it like a macro. Thus, the footnote renders.

Diversions and traps make the text formatting process non-linear. Let us imagine a set of text lines or paragraphs labelled '`A`', '`B`', and so on. If we set up a trap that produces text '`T`' (as a page footer, say), and we also use a diversion to store the formatted text '`D`', then a document with input text in the order '`A B C D E F`' might render as '`A B C E T F`'. The diversion '`D`' is never output if we do not call it.

Environments of themselves are not a source of non-linearity in document formatting: environment switches have immediate effect. One could always write a macro to change as many formatting parameters as desired with a single convenient call. But because diversions can be nested and macros called by traps that are sprung by other trap-called macros, they may be interpolated in varying contexts. For example, consider a page header that is always to be set in Helvetica. A document that uses Times for most of its body text, but Courier for displayed code examples, poses a challenge if a page break occurs in the middle of a code display; if the header trap assumes that the "previous font" is always Times, the rest of the example will be formatted in the wrong typeface. One could carefully save all formatting parameters upon entering the trap and restore them upon leaving it, but this is verbose, error-prone, and not future-proof as the `groff` language develops. Environments save us considerable effort.

5.29 Traps

Traps are locations in the output, or conditions on the input that, when reached or fulfilled, call a specified macro. These traps can occur at a given location either on the page or in the current diversion (together, these are known as *vertical position traps*), at a blank line, at a line with leading space characters, after a quantity of input lines, or at the end of input. Setting a trap is also called *planting* one. It is said that a trap is *sprung* if its condition is fulfilled. The formatter passes no arguments to macros called by traps.

5.29.1 Vertical Position Traps

A *vertical position trap* calls a macro when the formatter's vertical drawing position reaches or passes, in the downward direction, a certain location on the output page or in a diversion. Its applications include setting page headers and footers, body text in multiple columns, and footnotes.

`.vpt` [*b*] [Request]
`\n[.vpt]` [Register]

Enable or disable vertical position traps per Boolean expression *b*. They are enabled by default, and if *b* is omitted. Vertical position traps are those set by the `ch`, `wh`, and `dt` requests. Vertical position trap enablement is global. The read-only register `.vpt` stores its status.

A page can't be ejected if vertical position traps are disabled.[255]

5.29.1.1 Page Location Traps

A *page location trap* is a vertical position trap that applies to the page; that is, to the top-level diversion. Many can be present; manage them with the `wh` and `ch` requests.

`.wh` *dist* [*name*] [Request]

Plant macro *name* as page location trap at *dist*. The default scaling unit is '`v`'. Non-negative values for *dist* set the trap relative to the top of the page; negative values set the trap relative to the bottom of the page. It is not possible to plant a trap less than one basic unit from the page bottom: the formatter interprets a *dist* of `-0` as `0`, the top of the page. `wh` removes any existing *visible* trap (see below) at *dist*; this is its sole function if *name* is missing.

A trap springs only if it is *visible*, meaning that its location is reachable on the page[256] and it is not hidden by another trap at the same location already planted there.

[255] See Section 5.29.1.2 [The Implicit Page Trap], page 211.

[256] A trap planted at '`20i`' or '`-30i`' cannot spring on a page of length '`11i`'.

A macro package might set headers and footers as follows; this example configures vertical margins of one inch to the body text, and one half-inch to the titles. Observe the use of the no-break control character with the sp and bp requests to position our text baselines and prevent a partially collected line from being written outside the body text, and the page number character '%' used with the tl request.

```
.\" hdfo.roff
.de hd                  \" page header
'  sp .5i
.  tl '\\*(Ti''\\*(Da'  \" title and date strings
'  sp |1i
..
.de fo                  \" page footer
'  sp .5i-1v
.  tl ''%''
'  bp
..
.wh 0   hd              \" trap at top of the page
.wh -1i fo              \" trap 1 inch from bottom
```

Caution: A word about measurements is in order. Recall that the sp request vertically spaces such that the next text baseline (of one vee in height by definition) sets with the amount of space given to sp's argument *above* it. Thus in the example above, when the 'hd' trap springs at vertical position '0', invoking 'sp .5i', we get the desired half-inch of top margin. With the 'ft' trap, we space after the body text by one half-inch *minus one vee* to leave a half-inch bottom margin. The footer title, if taller than a baseline rule, thus "encroaches" into the half-inch margin between the body text and the bottom margin, just as the header title symmetrically intrudes into the half-inch of space between its own cap-height and that of the top of the body text.

To use these traps, copy the above (or load it from a file with the so or mso requests), then set up the strings it uses.

```
.so hdfo.roff
.ds Ti Final Report\"
.ds Da 21 May 2023\"
.ti
On 5 August of last year,
this committee tasked me with the investigation of the
CFIT (controlled flight into terrain) incident of
.\" ...and so on...
```

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. A negative trap value always uses the *current* page length; the formatter does not convert it to an absolute vertical position. We can use the pwh request to dump page

location traps to the standard error stream (see Section 5.38 [Debugging], page 240). GNU `troff` reports their positions in basic units, and includes empty slots in the list, where a trap had been planted but subsequently (re)moved, because they can affect the visibility of subsequently planted traps. An `nroff` device example follows.

```
.pl 5i
.wh -1i xx
.pwh
    error  xx      -240
.pl 100i
.pwh
    error  xx      -240
```

It is possible to have more than one trap at the same location (although only one at a time can be visible); to achieve this, the traps must be defined at different locations, then moved to the same place with the `ch` request. In the following example, the many empty lines caused by the `bp` request are not shown in the output.

```
.de a
.  nop a
..
.de b
.  nop b
..
.de c
.  nop c
..
.
.wh 1i a
.wh 2i b
.wh 3i c
.bp
    ⇒ a b c

.ch b 1i
.ch c 1i
.bp
    ⇒ a

.ch a 0.5i
.bp
    ⇒ a b
```

`\n[.t]` [Register]

The read-only register `.t` holds the distance to the next vertical position trap. If no such traps exist between the drawing position and the bottom of the page, it contains the distance to the page bottom. Within a

diversion, in the absence of a pending diversion trap, this distance is the maximum possible vertical position supported by the output device.

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

Change the location of a trap by moving macro *name* to new location *dist*, or by unplanting it altogether if *dist* is absent. The default scaling unit is 'v'. Parameters to `ch` are specified in the opposite order from `wh`. If *name* is the earliest planted macro of multiple traps at the same location, (re)moving it from that location exposes the macro next least recently planted at the same place.[257]

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. Blank lines have been elided from the output.

```
.de T
Trap sprung at \\n(nlu.
.br
..
.wh 1i T
.wh 2i T
foo
.sp 11i
.bp
.ch T 4i
bar
.sp 11i
.bp
.ch T 5i
baz
.sp 11i
.bp
.wh 5i
.ch T 6i
qux
.sp 11i
```

[257] 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.

```
⇒ foo
⇒ Trap sprung at 240u.
⇒ Trap sprung at 480u.
⇒ bar
⇒ Trap sprung at 480u.
⇒ Trap sprung at 960u.
⇒ baz
⇒ Trap sprung at 480u.
⇒ Trap sprung at 1200u.
⇒ qux
⇒ Trap sprung at 1440u.
```

`\n[.ne]` [Register]

The read-only register `.ne` contains the amount of vertical space demanded by the most recent `ne` request that sprang a vertical position trap; it is useful in conjunction with the `.trunc` register; recall Section 5.18 [Page Control], page 137. Since the `.ne` register is set only by traps, it doesn't make sense to interpolate it outside of macros called by traps.

`\n[.trunc]` [Register]

The read-only register `.trunc` contains the amount of vertical space truncated from an `sp` request's argument by the most recently sprung vertical position trap, or, if the trap was sprung by an `ne` request, minus the amount of vertical motion produced by the `ne` request. In other words, at the point a trap is sprung, it represents the difference of what the vertical position would have been but for the trap, and what the vertical position actually is. Since the `.trunc` register is set only by traps, it doesn't make sense to interpolate it outside of macros called by traps.

`\n[.trap]` [Register]

This read-only, string-valued register interpolates the name of the next vertical position trap that will be sprung.

`\n[.pe]` [Register]

This Boolean-valued, read-only register interpolates 1 while a page is being ejected, and 0 otherwise.

In the following example, we plant the same trap at the top and the bottom of the page. We also make the trap report its name and the vertical drawing position.

```
.de T
.tm \\$0: page \\n%, nl=\\n[nl] .pe=\\n[.pe]
..
.ll 46n
.wh 0 T
.wh -1v T
Those who can make you believe absurdities can make you
commit atrocities. \[em] Voltaire
    error  T: page 1, nl=0 .pe=0
    error  T: page 1, nl=2600 .pe=1
    ⇒ Those who can make you believe absurdities can
    ⇒ make you commit atrocities. -- Voltaire
```

When designing macros, keep in mind that diversions and traps do not normally interact. For example, if a trap calls a header macro (while outputting a diversion) that tries to change the font on the current page, the effect is not visible before the diversion has completely been printed (except for input protected with `\!` or `\?`) since the data in the diversion is already formatted. In most cases, this is not the expected behaviour.

5.29.1.2 The Implicit Page Trap

If, after starting GNU `troff` without loading a macro package, you use the `pwh` request to dump a list of the active traps to the standard error stream,[258] nothing is reported. Yet the `.t` register will report a steadily decreasing value with every output line your document produces, and once the value of `.t` gets to within `.V` of zero, you will notice that something trap-like happens—the page is ejected, a new one begins, and the value of `.t` becomes large once more.

This *implicit page trap* always exists in the top-level diversion;[259] its purpose is to eject the current page and start the next one. It works like a trap in some ways but not others. It has no name, so it cannot be moved or deleted with `wh` or `ch` requests. You cannot hide it by placing another trap at its location, and can move it only by redefining the page length with `pl`. Its operation is suppressed when vertical page traps are disabled with GNU `troff`'s `vpt` request.

5.29.1.3 Diversion Traps

A diversion is not formatted in the context of a page, so it lacks page location traps; instead it can have a *diversion trap*. There can exist at most one such vertical position trap per diversion. Use the `pwh` request to dump the properties of a diversion trap to the standard error stream.[260]

---

[258] See Section 5.38 [Debugging], page 240.
[259] See Section 5.30 [Diversions], page 217.
[260] See Section 5.38 [Debugging], page 240.

`.dt` [*dist name*] [Request]

Set a trap calling *name* from *within* a diversion at location *dist*, which is interpreted relative to diversion rather than page boundaries. If invoked with fewer than two arguments, any diversion trap in the current diversion is removed. The register `.t` works within diversions. It is an error to invoke `dt` in the top-level diversion. See Section 5.30 [Diversions], page 217.

5.29.2 Input Line Traps

`.it` [*n name*] [Request]
`.itc` [*n name*] [Request]
`\n[.it]` [Register]
`\n[.itc]` [Register]
`\n[.itm]` [Register]

Set an input line trap, calling macro *name* after processing the next *n* productive input lines (recall Section 5.9 [Manipulating Filling and Adjustment], page 106). Any existing input line trap in the environment is replaced. Without arguments, `it` and `itc` clear any input line trap that has not yet sprung.

Consider a macro '`.ST` *s n*' which sets the next *n* input lines in the font style *s*.

```
.de ST \" Use style $1 for next $2 text lines.
.  it \\$2 ES
.  ft \\$1
..
.de ES \" end ST
.  ft R
..
.ST I 1
oblique
face
.ST I 1
oblique\c
face
```

⇒ *oblique* face *obliqueface* (second "face" upright)

Unlike the `ce` and `rj` requests, `it` counts lines interrupted with the `\c` escape sequence separately (see Section 5.16 [Line Continuation], page 134); `itc` does not. To see the difference, let's change the previous example to use `itc` instead.

```
...
.  itc \\$2 ES
...
```

⇒ *oblique* face *obliqueface* (second "face" oblique)

You can think of the ce and rj requests as implicitly creating an input line trap with itc that schedules a break when the trap is sprung.

```
.de BR
.  br
.  internal: disable centering-without-filling
..
.
.de ce
.  if \\n[.br] .br
.  itc \\$1 BR
.  internal: enable centering-without-filling
..
```

The .it, .itc, and .itm registers respectively report the number of remaining productive input lines to be formatted before a pending input trap springs, a Boolean indication of whether that pending input trap honors output line continuation, and the name of the macro associated with the pending input trap. All are read-only; .itm is string-valued as well.

Let us consider in more detail the sorts of input lines that are or are not "productive".

```
.de Trap
TRAP SPRUNG
..
.de Mac
.if r a \l'5n'
..
.it 2 Trap
.
foo
.Mac
bar
baz
.it 1 Trap
.sp \" moves, but does not write or draw
qux
.itc 1 Trap
\h'5n'\c \" moves, but does not write or draw
jat
```

When 'Trap' gets called depends on whether the 'a' register is defined; the control line with the if request may or may not produce written output. We also see that the spacing request sp, while certainly affecting the output, does not spring the input line trap. Similarly, the horizontal motion escape sequence \h also affected the output, but was not "written". Observe that we had to follow it with \c and use itc to prevent

the newline at the end of the text line from causing a word break, which, like an ordinary space character, counts as written output.

```
$ groff -T ascii input-trap-example.groff
    ⇒ foo bar TRAP SPRUNG baz
    ⇒
    ⇒ qux TRAP SPRUNG      jat TRAP SPRUNG
$ groff -T ascii -r a1 input-trap-example.groff
    ⇒ foo _____ TRAP SPRUNG bar baz
    ⇒
    ⇒ qux TRAP SPRUNG      jat TRAP SPRUNG
```

Input line traps are associated with the environment (see Section 5.32 [Environments], page 224); switching to another environment suspends the current input line trap, and going back resumes it, restoring the count of qualifying lines enumerated in that environment.

5.29.3 Blank Line Traps

`.blm` [*name*] [Request]

Set a blank line trap, calling the macro *name* when GNU `troff` encounters a blank line in input, instead of the usual behavior (see Section 5.1.4 [Breaking], page 70). A line consisting only of spaces is also treated as blank and subject to this trap. Omitting the argument reëstablishes the default blank line behavior.

5.29.4 Leading Space Traps

`.lsm` [*name*] [Request]
`\n[lsn]` [Register]
`\n[lss]` [Register]

Set a leading space trap, calling the macro *name* when GNU `troff` encounters leading spaces on a text line; the implicit line break that normally happens in this case is suppressed. The formatter stores the count of leading spaces on the text line 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, GNU `troff` removes the leading spaces from the input line and produces no motion before calling *name*.

Omitting the argument reëstablishes the default handling of leading spaces on text lines (breaking the line when filling, and formatting a horizontal motion of '`\n[lsn]`' word spaces).

5.29.5 End-of-input Traps

`.em` [*name*] [Request]

Set a trap at the end of input; the formatter calls macro *name* after processing the last line of the last input file. Without an argument, the formatter removes any existing end-of-input trap.

For example, if the document had to have a section at the bottom of the last page for someone to approve it, the em request could be used.

```
.de approval
\c
.  ne 3v
.  sp (\\n[.t]u - 3v)
.  in +4i
.  lc _
.  br
Approved:\t\a
.  sp
Date:\t\t\a
..
.
.em approval
```

The \c in the above example needs explanation. For historical reasons (compatibility with AT&T troff), the end-of-input macro exits as soon as it causes a page break if no partially collected line remains.[261]

Let us assume that there is no \c in the above approval macro, that the page is full, and last output line has been broken with, say, a br request. Because there is no more room, a ne request at this point causes a page ejection, which in turn makes troff exit immediately as just described. In most situations, this is not desired; people generally want to format the input after ne.

To force processing of the whole end-of-input macro independently of this behavior, it is thus advisable to (invisibly) ensure the existence of a partially collected line (\c) whenever there is a chance that a page break can happen. In the above example, invoking the ne request ensures that there is room for the subsequent formatted output on the same page, so we need insert \c only once.

The next example shows how to append three lines, then start a new page unconditionally. Since '.ne 1' doesn't give the desired effect—there is always one line available or we are already at the beginning of the next page—we temporarily increase the page length by one line so that we can use '.ne 2'.

[261] While processing an end-of-input macro, the formatter assumes that the next page break must be the last; it goes into "sudden death overtime".

```
.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 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 \n[.R]u
.  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.

5.30 Diversions

In roff systems it is possible to format text as if for output, but instead of writing it immediately, one can *divert* the formatted text into a named storage area. It is retrieved later by specifying its name after a control character. The formatter uses the same name space for such *diversions* as for strings and macros; recall Section 5.5 [Identifiers], page 86. Such text is sometimes said to be "stored in a macro", but this coinage obscures the important distinction between macros and strings on one hand and diversions on the other; the former store *unformatted* input text, and the latter capture *formatted* output.[262] Diversions also do not interpret arguments. Applications of diversions include footnotes, tables of contents, indices, and "keeps" (preventing a page break from occurring at an inconvenient place by forcing a set of output lines to be set as a group). For orthogonality it is said that GNU troff populates the *top-level diversion* if no diversion is active (that is, formatted output is being "diverted" directly to the output device). The top-level diversion has no name.

Dereferencing an undefined diversion creates an empty one of that name.[263] A diversion does not exist for the purpose of testing with the d conditional expression operator until its initial definition ends; recall Section 5.23.1 [Operators in Conditionals], page 175. The following requests create and alter diversions.

.di [*name*] [Request]
.da [*name*] [Request]

Start collecting formatted output in a diversion called *name*. The da request appends to a diversion called *name*, creating it if necessary. If *name* already exists as an alias, the target of the alias is replaced or appended to; recall Section 5.22 [Strings], page 170. The pending output line is diverted as well. Switching to another environment (with the ev request) before invoking di or da avoids including any pending output line in the diversion.[264]

Invoking di or da without an argument stops diverting output to the diversion named by the most recent corresponding request. Invoking di or da without an argument when no diversion is being populated does nothing.[265]

---

262 See Section 5.37 [GNU troff Internals], page 237.

263 GNU troff emits a warning in category 'mac'. See Section 5.38.1 [Warnings], page 245.

264 See Section 5.32 [Environments], page 224.

265 GNU troff emits a warning in category 'di'. See Section 5.38.1 [Warnings], page 245.

```
.ll 56n
Ahoy, me hearties,
I traveled unto a distant isle,
.br
.di HT
and thereupon I lay a vast treasure,
.br
.di
.HT
.br
which none o' ye shall ever see.
    ⇒ Ahoy, mateys, I traveled unto a distant isle,
    ⇒ and thereupon I lay a vast treasure,
    ⇒ which none o' ye shall ever see.
```

GNU `troff` supports *box* requests to exclude a partially collected line from a diversion, as this is often desirable.

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

Divert (or append) output to *name*, similarly to the `di` and `da` requests, respectively. Any pending output line is *not* included in the diversion. Without an argument, stop diverting output; any pending output line inside the diversion is discarded.

```
.ll 56n
Ahoy, mateys,
I traveled unto a distant isle,
.br
.box SECRET
and thereupon I lay a vast treasure,
.br
accurst wi' neutron activation,
.box
.SECRET
.br
which none o' ye shall ever see.
    ⇒ Ahoy, mateys, I traveled unto a distant isle,
    ⇒ and thereupon I lay a vast treasure,
    ⇒ which none o' ye shall ever see.
```

Apart from pending output line inclusion and the request names that populate them, boxes are handled exactly as diversions are. All of the following `groff` language elements can be used with them interchangeably.

`\n[.z]` [Register]
`\n[.d]` [Register]

Diversion requests may be nested. The read-only string-valued register `.z` contains the name of the current diversion. The read-only register `.d`

contains the vertical drawing position in the diversion. If the input text is not being diverted, .d reports the same location as the register nl.

```
.nf
.di A
alpha
.di B
beta
.di
gamma
\*B
.di
delta
\*A
epsilon
    ⇒ delta
    ⇒ alpha
    ⇒ gamma
    ⇒ beta
    ⇒
    ⇒
    ⇒ epsilon
```

\n[.h] [Register]

The read-only register .h stores the *high-water mark* on the current page or in the current diversion. It corresponds to the text baseline of the lowest line on the page.[266]

```
.tm .h==\n[.h], nl==\n[nl]
    ⇒ .h==0, nl==-1
This is a test.
.br
.sp 2
.tm .h==\n[.h], nl==\n[nl]
    ⇒ .h==40, nl==120
```

As implied by the example, vertical motion does not produce text baselines and thus does not increase the value interpolated by '\n[.h]'.

\n[dn] [Register]
\n[dl] [Register]

After output to a (named) diversion stops, the formatter stores its vertical and horizontal sizes, to the writable registers dn and dl, respectively. Only the lines just processed are counted: for the computation of dn and dl, the requests da and boxa are handled as if di and box had been used, respectively—lines that have been already stored in the diversion (box) are not taken into account.

[266] Thus, the "water" gets "higher" proceeding *down* the page.

```
.\" Center text both horizontally and vertically.
.\" Macro .(c starts centering mode; .)c terminates it.
.
.\" Disable the escape character with .eo so that we
.\" don't have to double backslashes on the "\n"s.
.eo
.de (c
.  br
.  ev (c
.  evc 0
.  in 0
.  nf
.  di @c
..

.de )c
.  br
.  ev
.  di
.  nr @s (((\n[.t]u - \n[dn]u) / 2u) - 1v)
.  sp \n[@s]u
.  ce 1000
.  @c
.  ce 0
.  sp \n[@s]u
.  br
.  fi
.  rr @s
.  rm @c
..
.ec
```

\!*character-sequence* [Escape sequence]
\?*character-sequence*\? [Escape sequence]

Transparently embed *character-sequence* into the current diversion, preventing the formatter from interpreting requests, macro calls, and escape sequences when reading them into a diversion. Doing so prevents them from taking effect until the diverted text is actually output. The `\!` escape sequence transparently embeds input up to and including the end of the line. The `\?` escape sequence transparently embeds input, read in copy mode, up to its own next occurrence on the input line. Use `\!` by itself to embed newlines in a diversion. The two escape sequences differ in that GNU `troff` interprets `\?` even in copy mode; recall Section 5.24.2 [Copy Mode], page 188. Consequently, comparands protected with `\?` need not be valid GNU `troff` syntax.

```
.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
    ⇒ 4
```

Both escape sequences read the data in copy mode.

If `\!` is used in the top-level diversion, its argument is embedded into GNU `troff`'s device-independent output. One of its applications is control of a postprocessor that transforms the data that are subsequently read by an output driver.

Using the `\?` escape sequence in the top-level diversion produces no output at all; its argument is simply ignored.

`.output` ["]*character-sequence* [Request]

Emit *character-sequence* directly to GNU `troff`'s output; this usage is similar to that of `\!` when it occurs in the top-level diversion.

GNU `troff` removes a leading neutral double quote '`"`' from *character-sequence*, permitting initial embedded spaces in it, and reads it to the end of the input line in copy mode. Recall Section 5.24.2 [Copy Mode], page 188.

Caution: Use of these features can put syntactically invalid content into the formatter's output, which `groff`'s output drivers then fail to process. One application of `output` and of `\!` from the top-level diversion is to pass instructions to a postprocessor that interprets *character-sequence* and filters it out before sending it to the output driver. If all you require is to encode character code points that are not meaningful in `roff` device-independent output but are to an output driver, such as Unicode characters in document metadata, bookmarks, or URLs, consider the `device` request or `\X` escape sequence.[267]

`.asciify` *div* [Request]

Unformat the diversion *div* in a way such that Unicode basic Latin (US-ASCII) characters, characters translated with the `trin` request, space

[267] See Section 5.35 [Postprocessor Access], page 235.

characters, and some escape sequences that were formatted and diverted into *div* are treated like ordinary input characters when *div* is interpolated. Doing so can be useful in conjunction with the `writem` request.

When transforming a glyph node back into an input sequence that demands expression as a special character escape sequence, GNU `troff` uses the default escape character.

`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 nodes in a diversion to their source equivalents: those produced by indexed characters (`\N`), for example, remain nodes, so the result cannot be guaranteed to be a character sequence as a macro or string is. Give the diversion name as an argument to the `pm` request to inspect its contents and node list. Glyph parameters such as the type face and size are not preserved; use `unformat` to achieve that.

`.unformat` *div* [Request]

Like `asciify`, unformat the diversion *div*. However, `unformat` handles only tabs and spaces between words, the latter usually arising from spaces or newlines in the input. Tabs are treated as tokens, and spaces become adjustable again. The vertical sizes of lines are not preserved, but glyph information (font, type size, space width, and so on) is retained.

5.31 Punning Names

Macros, strings, and diversions share a name space; recall Section 5.5 [Identifiers], page 86. Internally, the same mechanism is used to store them. You can thus call a macro with string interpolation syntax and vice versa.

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

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

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

By ending each text line of the macros with an escaped RET, we get the desired effect; recall Section 5.16 [Line Continuation], page 134.[268] What would have happened if we had used only one backslash in each case?

Interpolating a string does not hide existing macro arguments. We can also place the escaped newline outside the string interpolation instead of within the string definition. Thus, in a macro, a more efficient way of doing

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

is

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

The latter calling syntax doesn't change the value of \$0, which is then inherited from the calling macro; recall Section 5.24.1 [Parameters], page 186.

It is sometimes convenient to copy a single-line diversion to a string, which can then be interpolated with *.

```
.di xx
the
.ft I
interpolation system
.ft
.br
.di
.ds yy This is a test of \*(xx\c
\*(yy.
    ⇒ This is a test of the interpolation system.
```

In the foregoing, we see that formatted output can thus be stored in a string. The \c escape sequence prevents the subsequent newline from being interpreted as a break; again, recall Section 5.16 [Line Continuation], page 134.

Copying multi-output-line diversions produces unexpected results.

[268] We must double the backslash. Recall Section 5.24.2 [Copy Mode], page 188.

```
.di xxx
a funny
.br
test
.br
.di
.ds yyy This is \*[xxx]\c
\*[yyy].
    ⇒ test This is a funny.
```

Usually, it is not predictable whether a diversion contains one or more output lines, so this mechanism should be avoided. With AT&T `troff`, this was the only solution to strip off a final newline from a diversion. Another disadvantage is that the spaces in the copied string are already formatted, preventing their adjustment. This can cause ugly results.

A clean solution to this problem is available in GNU `troff`, using the requests `chop` to remove the final newline of a diversion, and `unformat` to make the horizontal spaces adjustable again.

```
.box xxx
a funny
.br
test
.br
.box
.chop xxx
.unformat xxx
This is \*[xxx].
    ⇒ This is a funny test.
```

See Section 5.37 [GNU `troff` Internals], page 237.

5.32 Environments

As discussed in Section 5.28 [Deferring Output], page 204, environments store most of the parameters that determine the appearance of text. A default environment named '`0`' (zero) exists when the formatter starts up; formatting-related requests and escape sequences modify its properties.

You can create new environments and switch among them. Only one is current at any given time. Active environments are managed using a *stack*, a data structure supporting "push" and "pop" operations. The current environment is at the top of the stack. The same environment name can be pushed onto the stack multiple times, possibly interleaved with others. Popping the environment stack does not destroy the current environment; it remains accessible by name and can be made current again by pushing it at any time. Environments cannot be renamed or deleted, and can only

be modified when current. To inspect the environment stack, use the pev request.[269]

Environments store the following information.

- a partially collected line, if any
- data about the most recently output glyph and line (registers .cdp, .cht, .csk, .n, .w)
- typeface parameters (type size—selected and requested, default family, resolved font selection, height and slant, inter-word and inter-sentence space sizes; and previous values of type size—selected and requested, default family, resolved font selection)
- page parameters (line length, title length, vertical spacing, post-vertical spacing line spacing, indentation—normal and temporary, line numbering, centering, right-alignment, underlining, hyphenation parameters; and previous values of line length, title length, vertical spacing, post-vertical spacing, line spacing, indentation—normal and temporary)
- filling enablement; adjustment enablement and mode
- tab stops; tab, leader, escape, control, no-break control, hyphenation, and margin characters
- input line traps
- stroke and fill colors, selected and previous

.ev [*ident*] [Request]
\n[.ev] [Register]

Enter the environment *ident*, which is created if it does not already exist, using the same parameters as for the default environment used at startup. With no argument, GNU troff switches to the previous environment.

Invoking ev with an argument puts environment *ident* onto the top of the environment stack. (If it isn't already present in the stack, this is a proper push.) Without an argument, ev pops the environment stack, making the previous environment current. It is an error to pop the environment stack with no previous environment available. The read-only string-valued register .ev contains the name of the current environment—the one at the top of the stack.

[269] See Section 5.38 [Debugging], page 240.

```
.ev footnote-env
.fam N
.ps 6
.vs 8
.ll -.5i
.ev

...

.ev footnote-env
\[dg] Observe the smaller text and vertical spacing.
.ev
```

We can familiarize ourselves with stack behavior by wrapping the `ev` request with a macro that reports the contents of the `.ev` register to the standard error stream.

```
.de EV
.  ev \\$1
.  tm environment is now \\n[.ev]
..
.
.EV foo
.EV bar
.EV
.EV baz
.EV
.EV
.EV
```

```
error  environment is now foo
error  environment is now bar
error  environment is now foo
error  environment is now baz
error  environment is now foo
error  environment is now 0
error  error: environment stack underflow
error  environment is now 0
```

`.evc` *environment* [Request]

Copy the properties of *environment* to the current environment, except for:

- a partially collected line, if present;
- the interruption status of the previous input line (due to use of the `\c` escape sequence);
- the count of remaining lines to center, to right-justify, or to underline (with or without underlined spaces)—these are set to zero;
- the activation status of temporary indentation;

- input line traps and their associated data;
- the activation status of line numbering (which can be reactivated with '.nm +0'); and
- the count of consecutive hyphenated lines (set to zero).

Copying an environment to itself discards the foregoing data.

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

The \n[.w] read-only register contains the width of the last glyph formatted in the environment.[270]

The \n[.cht] read-only register contains the height of the last glyph formatted in the environment.

The \n[.cdp] read-only register contains the depth of the last glyph formatted in the environment. It is positive for glyphs extending below the baseline.

The \n[.csk] read-only register contains the *skew* (how far to the right of the glyph's center that GNU troff should place an accent) of the last glyph formatted in the environment.

\n[.n] [Register]

The \n[.n] read-only register contains the length of the output line most recently formatted in the environment.

5.33 Suppressing Output

\O[*num*] [Escape sequence]

Suppress GNU troff output of glyphs and geometric objects. The sequences \O2, \O3, \O4, and \O5 are intended for internal use by grohtml.

'\O0' Disable the emission of glyphs and geometric objects to the output driver, provided that this sequence occurs at the outermost suppression level (see \O3 and \O4 below). Horizontal motions corresponding to non-overstruck glyph widths still occur.

'\O1' Enable the emission of glyphs and geometric objects to the output driver, provided that this sequence occurs at the outermost suppression level.

\O0 and \O1 also reset the four registers opminx, opminy, opmaxx, and opmaxy to −1. These four registers mark the top left and bottom right hand corners of a box encompassing all written or drawn output.

[270] This quantity is also known as the *advance width*; recall the cs request description in [Artificial Fonts], page 156.

'\O2' At the outermost suppression level, enable emission of glyphs and geometric objects, and write to the standard error stream the page number and values of the four aforementioned registers encompassing glyphs written since the last interpolation of a \O sequence, as well as the page offset, line length, image file name (if any), horizontal and vertical device motion quanta, and input file name. Numeric values are in basic units.

'\O3' Begin a nested suppression level. grohtml uses this mechanism to create images of output preprocessed with pic, eqn, and tbl. At startup, GNU troff is at the outermost suppression level. pre-grohtml generates these sequences when processing the document, using GNU troff with the ps output device, Ghostscript, and the PNM tools to produce images in PNG format. They start a new page if the device is not html or xhtml, to reduce the number of images crossing a page boundary.

'\O4' End a nested suppression level.

'\O[5*Pfile*]'
At the outermost suppression level, write the name file to the standard error stream at position *P*, which must be one of l, r, c, or i, corresponding to left, right, centered, and inline alignments within the document, respectively. *file* is a name associated with the production of the next image.

\n[.O] [Register]
The output suppression nesting level applied by \O3 and \O4 is available in the read-only register '.O'.

5.34 Host System Service Access

Occasionally a document wants to access the system clock, file storage, or other services provided by the operating environment.

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

The formatter sets date- and time-related registers per the local time as determined by *localtime*(3) when the formatter launches. This initialization can be influenced by the environment variable TZ and overridden by SOURCE_DATE_EPOCH; recall Section 2.2 [Environment], page 13.

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

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

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

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

\n[dy] Day of the month (1–31).

\n[mo] Month of the year (1–12).

\n[year] Gregorian year.

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

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

can be corrected to:

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

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

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

If you wish to embed the date and/or time of a document's formatting into its output, interpolate these registers into its text. Use the af request to format their values for output.

```
.af year 0000
.af mo 00
.af dy 00
.af hours 00
.af minutes 00
.af seconds 00
ISO 8601 date stamp:
\n[year]-\n[mo]-\n[dy]T\n[hours]:\n[minutes]:\n[seconds]
    ⇒ ISO 8601 date stamp: 2025-12-07T02:17:54
```

roff formatters allow files to be read into the input stream. Enabling GNU troff's unsafe mode at the command line permits the writing of files and execution of external commands, with or without inclusion of their output in the document.

Caution: The requests discussed below that accept a file name or system command as an argument treat the remainder of the input line as that argument, including any spaces, up to a newline or comment escape sequence. Suffixing the file name or command with a comment, even an empty one, prevents unwanted space from creeping into it during source document maintenance. GNU troff removes a leading neutral double quote '"' from such

an argument, permitting initial embedded spaces in it, and reads it to the end of the input line in copy mode. Recall Section 5.24.2 [Copy Mode], page 188.

`.so` ["]*file* [Request]
`.soquiet` ["]*file* [Request]

The `so` request replaces its own control line with the contents of *file*, "sourcing" it: the formatter then interprets the contents of *file* as input. GNU `troff` searches for *file* in any directories specified by `-I` command-line options, followed by the current working directory. If *file* does not exist, the formatter ignores the request.[271]

`so` can be useful for large documents, allowing each chapter of a book, for example, to be maintained in a separate file. However, files interpolated with `so` are not preprocessed; to overcome this limitation, see *soelim*(1).

Caution: Since the formatter replaces the entire control line with the contents of a file, *file* must end with a newline, or the formatter will continue reading the next input line of the `roff` file as if it were part of the last line of the sourced file. Consider a file `xxx` containing only the word '`foo`' without a trailing newline.

```
$ printf 'foo' > xxx
$ groff -T ascii <<EOF
The situation is
.so xxx
bar.
EOF
    ⇒ The situation is foobar.
```

`soquiet` works the same way, except that GNU `troff` issues no warning diagnostic if *file* does not exist.

`.pso` ["]*command* [Request]

Read the standard output from the specified *command* when passed to *popen*(3) and include it in place of the `pso` request.

It is an error to use this request in safer mode, which is the default. Invoke GNU `troff` or a front end with the `-U` option to enable unsafe mode.

The cautionary note regarding a final newline in the stream read by the `so` request applies to `pso` as well.

`.mso` ["]*file* [Request]
`.msoquiet` ["]*file* [Request]

As the `so` and `soquiet` requests, respectively, except that GNU `troff` searches for the specified *file* in the same directories as macro files; recall `GROFF_TMAC_PATH` in Section 2.2 [Environment], page 13, and `-m` in Section 2.1 [Groff Options], page 7.

[271] GNU `troff` emits a warning in category '`file`'. See Section 5.38.1 [Warnings], page 245.

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

Break and copy the contents of *file* as "throughput" to GNU troff's output. Each line of *file* is output as if preceded by \!, but is not interpreted by the formatter. If *file* does not end with a newline, trf appends one. Both requests break the line before reading *file*, unless invoked with the no-break control character. If a diversion is in use, GNU troff performs the copy only when the diversion is emitted.

cf copies the contents of *file* completely unprocessed; it is therefore an error to use this request in safer mode, which is the default. Invoke GNU troff or a front end with the -U option to enable unsafe mode.

trf discards input characters that are invalid in GNU troff's output format, meaning all character codes less than decimal 32 (except 10, the line feed) and greater than decimal 127.[272]

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 \!\\!.

Caution: Use of these requests can put syntactically invalid content into the formatter's output, which groff's output drivers then fail to process. One application is to pass instructions to a postprocessor that interprets *file*'s contents and filters it out before sending it to the output driver.

You can define a diversion x containing the contents of file f as follows.

```
.ev 1
.di x
.trf f
.di
.ev
```

Switching environments prevents any existing partially collected output line from becoming part of the diversion; recall Section 5.30 [Diversions], page 217.

You could use a box diversion to achieve the same end, but if filling is enabled you must ensure that the output line breaks before closing the diversion.

```
.box x
.trf f
.br
.box
```

In AT&T troff, cf copies the contents of *file* to the output immediately even if a diversion is open; this behavior is so anomalous that we consider it a bug.

.nx [["]*file*] [Request]

Stop processing the input file. If *file* is specified, read it; otherwise, read the next pending input file, if any.

[272] See Section 6.2 [GNU troff Output], page 263.

`.rd` [*prompt* [*arg1 arg2* . . .]] [Request]

Read from standard input, and include what is read as though it were part of the input file. Text is read until a blank line is encountered.

If standard input is a TTY input device (keyboard), write *prompt* to standard error, followed by a colon (or send BEL for a beep if no argument is given).

Arguments after *prompt* are available for the input. For example, the line

```
.rd data foo bar
```

with the input '`This is \$2.`' prints

```
This is bar.
```

Using the `nx` and `rd` requests, it is easy to set up form letters. The form letter template is constructed like this, putting the following lines into a file called `repeat.let`:

```
.ce
\*(td
.sp 2
.nf
.rd
.sp
.rd
.fi
Body of letter.
.bp
.nx repeat.let
```

When the foregoing runs, redirect standard input from a file like that shown below to `troff`. The formatter executes any requests it contains as though they were part of the form letter. The last line of the file is an `ex` request, which tells the formatter to stop processing. If it were absent, the formatter would hang awaiting further input.

```
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` ["]*command* [Request]

Use the formatter's device-independent output as the input to the commands specified in *pipe* and emit their output to the standard output stream instead of the formatter's usual output. The formatter reads the remainder of the input line into *command* and passes it to *popen*(3). The formatter does not capture output produced by the command(s).

Multiple `pi` requests construct a multi-stage pipeline in the same order as the formatter encounters the requests. Thus,

```
.pi foo
.pi bar
```

is the same as '`.pi foo | bar`'.

`pi` must be invoked before GNU `troff` writes any nodes to its output.[273] The formatter reports an error and ignores the request if `pi` is invoked "too late". Roughly, this means you should set up your `pi` pipeline early in a document, before anything but register, string, and macro definitions (and/or sourcing of files that comprise these exclusively).

Use of this request in safer mode (GNU `troff`'s default) is erroneous. Invoke GNU `troff` or a front end with the `-U` option to enable unsafe mode.

Caution: Use of the `pi` request can put syntactically invalid content into the formatter's output, which `groff`'s output drivers then fail to process. The pipeline you construct is responsible for maintaining the validity of the input to the output driver.

`.sy` ["]*command* [Request]
`\n[systat]` [Register]

The `sy` request executes the specified shell command(s). The formatter reads the remainder of the input line into a buffer, and passes it to *system*(3) without capturing the output produced by the command(s).

It is an error to use this request in safer mode; this is the default. Give GNU `troff` or a front end program the `-U` command-line option to enable unsafe mode.

The writable register `systat` contains the return value of the *system*(3) function executed by the most recent `sy` request.

Real-world (and non-malicious) applications of `sy` are esoteric; the request interpolates neither the standard output nor the standard error streams of *command* into the document. Worse, AT&T `troff` afforded no means of verifying that *command* operated as expected—thus the GNU `systat` register extension. We therefore offer a silly example of its use, making a document refuse to format if the system user name running the formatter is '`branden`'.[274]

---

[273] See Section 5.37 [GNU troff Internals], page 237.

[274] POSIX command environments and `roff` formatters employ different integer-to-Boolean interpretation conventions; a POSIX command exits with a zero status if

```
.ds user branden\"
.sy test "$(id -un)" = \*[user]
.if \n[systat]=0 .ab formatting refused for \*[user]
Hello, world!
```

`.open` *stream* ["]*file* [Request]
`.opena` *stream* ["]*file* [Request]

The `open` request opens *file* for writing and associates a stream named *ident* with it, making it available for `write` requests. The `opena` request is similar, but appends to *file* if it already exists instead of overwriting it. It is an error to use these requests in safer mode; this is the default. Give GNU `troff` or a front end program the `-U` command-line option to enable unsafe mode. The `pstream` request dumps the list of open streams to the standard error stream.[275]

`.write` *stream* ["]*contents* [Request]
`.writec` *stream* ["]*contents* [Request]

The `write` request writes *contents* and a newline to *stream*, which must previously have been the subject of an `open` (or `opena`) request. GNU `troff` flushes the stream after writing to it. The `writec` request is similar, but does not write a newline to *stream*.

`.writem` *stream name* [Request]

Write the contents of the macro or string *name* to *stream*, which must previously have been the subject of an `open` (or `opena`) request. GNU `troff` reads the contents of *name* in copy mode. That is, it ignores already formatted elements (nodes). Consequently, diversions must be unformatted with the `asciify` request before calling `writem`. Usually, this means a loss of information.

`.close` *stream* [Request]

Close the specified *stream*, making it unacceptable as an argument to the `write` request.

Here's how a document might arrange to construct an (unsorted) index for itself.

```
.open idx test.idx
.
.de IX
.  write idx \\n[%] \\$*
..
.
.IX test entry
.
.close idx
```

---

it succeeds and a positive one if it fails, whereas a `roff` register tests "true" if it has a positive value.

[275] See Section 5.38 [Debugging], page 240.

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

Interpolate the contents of the system environment variable *env* (one-character name *e*, two-character name *ev*) as returned by *getenv*(3). GNU troff interprets \V even in copy mode; recall Section 5.24.2 [Copy Mode], page 188.

5.35 Postprocessor Access

Beyond the cf and trf requests (recall Section 5.34 [Host System Service Access], page 228), two escape sequences and two requests enable documents to pass information directly to a postprocessor. These are useful for exercising device-specific capabilities that the groff language does not abstract or generalize; examples include the embedding of hyperlinks and image files. Device-specific functions are documented in each output driver's man page, such as *gropdf*(1), *grops*(1), or *grotty*(1).

.device ["]*character-sequence* [Request]
\X'*contents* . . . ' [Escape sequence]

Embed *character-sequence* into GNU troff output as parameters to an 'x X' device extension command.[276]

GNU troff removes a leading neutral double quote '"' from *contents*, permitting initial embedded spaces in it, and reads it to the end of the input line in copy mode. Recall Section 5.24.2 [Copy Mode], page 188.

The groff special character repertoire is unknown to output drivers outside of glyphs named in a device's fonts, and even then they may not possess complete coverage of the names documented in the *groff_char*(7) man page. Further, escape sequences that produce horizontal or vertical motions, hyphenation points, or that are dummy characters may appear in strings or be converted to nodes, particularly in diversions.[277] These are not representable when interpolated directly into device-independent output, as might be done when writing out tag names for PDF bookmarks, which can appear in a viewer's navigation pane.

So that documents or macro packages do not have to laboriously "sanitize" strings destined for interpolation in device extension commands, the \X escape sequence performs certain transformations on its argument. For these transformations, character translations and definitions are ignored.

GNU troff converts several ordinary characters that typeset as non-basic Latin code points to code points outside that range so that they are used consistently whether they are formatted as glyphs or used in a device extension command argument. These ordinary characters are '!', '-', '^', '`', and '~'; others are written as-is.

[276] See Section 6.2 [GNU troff Output], page 263.
[277] See Section 5.37 [GNU troff Internals], page 237.

Special characters that typeset as Unicode basic Latin characters are translated to basic Latin characters accordingly. So that any Unicode code point can be represented in a device extension command, for example in an author's name in document metadata or as a usefully named bookmark or hyperlink anchor, GNU `troff` maps other special characters to Unicode special character notation. Recall Section 5.19.4 [Characters and Glyphs], page 146.

GNU `troff` does not write special characters without a Unicode representation and escape sequences that do not interpolate a sequence of ordinary and/or special characters as arguments to device extension commands.[278]

To avoid the transformations `device` and `\X` perform, employ the `output` or `\!` or `\?` escape sequences; recall See Section 5.30 [Diversions], page 217.

GNU `troff` also permits the interpolation of macro or string contents as a device extension command.

`.devicem` *name* [Request]
`\Y`*n* [Escape sequence]
`\Y(`*nm* [Escape sequence]
`\Y[`*name*`]` [Escape sequence]

The `devicem` request and `\Y` escape sequence correspond to '`.device \*[`*name*`]`' and '`\X'\*[`*name*`]'`' (one-character name *n*, two-character name *nm*), respectively. They differ from their counterparts in that GNU `troff` does not interpret the contents of the string or macro *name*; further, *name* may be a macro and thus contain newlines. (There is no way to embed a newline in the arguments to `device` or `\X`.) The inclusion of newlines requires an extension to the AT&T `troff` device-independent page description language, and their presence confuses drivers that do not know about it.[279]

Use of device extension commands early in a document (before the first text is formatted) can interfere with processing of page location traps. If you experience problems when placing them early, precede the first with a dummy character escape sequence '`\&`'; recall Section 5.19.10 [Dummy Characters], page 160.

`.tag` *name* [Request]
`.taga` *name* [Request]

Reserved for internal use.

5.36 Miscellaneous

We document here GNU `troff` features that fit poorly elsewhere.

---

[278] When encountered, these produce warnings in category '`char`'. See Section 5.38.1 [Warnings], page 245.

[279] See Section 6.2.2.4 [Device Control Commands], page 271.

.psbb *file* [Request]
\n[llx] [Register]
\n[lly] [Register]
\n[urx] [Register]
\n[ury] [Register]

Retrieve the bounding box of the PostScript image found in *file*, which must conform to Adobe's *Document Structuring Conventions* (DSC), locate a %%BoundingBox comment, and store the (upper-, lower-, -left, -right) values into the registers llx, lly, urx, and ury. If an error occurs (for example, if no %%BoundingBox comment is present), the formatter sets these registers to 0.

Control the search path for *file* with the -I command-line option.

5.37 GNU troff Internals

GNU troff processes input in three steps. It gathers one or more input characters into a *token*,[280] the smallest meaningful unit of troff input. The process of formatting translates tokens into nodes that populate a pending output line (recall Section 5.9 [Manipulating Filling and Adjustment], page 106). A *node* is a data structure representing any object that may ultimately appear in the output, like a glyph or motion on the page. When the pending output line breaks, the formatter applies any relevant adjustment, line number, and margin character, and finally appends it to the current diversion. Periodically, the formatter *flushes* accumulated output line(s) to the output device, a process that translates each node into a device-independent output language representation understood by all output drivers. Copy mode tokenizes but does not format; diversions (apart from that at the top level) format but do not write output.

For example, GNU troff converts the input 'Gi\[:u]\%seppe' into a character token for 'g', a character token for 'i', a special character token for ':u' (representing 'u' with an umlaut), a token encoding a hyphenation point,[281] and further character tokens. You can observe this process by storing the foregoing input into a string—which, because its contents are read in copy mode, is only tokenized, not formatted—and dumping it with the pm request.[282] (Using *printf*(1) requires us to double the '\' and '%' characters.)

[280] When not in copy mode, the formatter does not tokenize the escape sequences \f, \F, \H, \m, \M, \R, \s, and \S, but instead updates its global register dictionary (\R), or the environment (all others).

[281] GNU troff encodes tokens that aren't Unicode Basic Latin characters as code points in the C0 and C1 control ranges; we plan to move them to the Unicode Private Use Area (PUA) or to code points outside the Unicode encoding space in a future release.

[282] Because GNU troff's internals are subject to revision, we do not show the output of these examples. The names and structures of node types may change over time. The JSON interpreter *jq*(1) is not essential, but can be helpful in understanding the topology of the node trees populating output lines and diversions in particular.

```
$ printf '.ds str Gi\\[:u]\\%%seppe\n.pm str\n' \
    | groff 2>&1 | jq
```

Similarly, we can observe the details of the formatting process by interpolating the string, or supplying its contents directly as input, and invoking the `pline` request.

```
$ printf 'Gi\\[:u]\\%%seppe\n.pline\n' | groff -z 2>&1 | jq
```

We now see a list of nodes, including an output line start node, several glyph nodes, a discretionary break node containing a glyph node for the special character '`:u`' *and* a glyph node for the special character '`hy`' (hyphen), and a word space node at the end corresponding to the newline at the end of input.[283]

If we change '`G`' to '`f`', we see that the first two glyph nodes, for '`f`' and '`i`', become contained by a ligature node (provided the current font has a glyph for this ligature). All output glyph nodes are "processed", which means that they are associated with a given font, type size, advance width, and so forth.

Macros, diversions, and strings collect elements in two chained lists: a list of tokens that have been passed unprocessed, and a list of nodes. Consider the following diversion.

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

It contains these elements.

node list	token list	element number
line start node	—	1
glyph node `a`	—	2
word space node	—	3
—	`b`	4
—	`\n`	5
glyph node `c`	—	6
vertical size node	—	7
vertical size node	—	8
—	`\n`	9

[283] You may wonder why a glyph node for '`hy`' exists when this example doesn't produce a hyphen in the output. That's because the hyphenation break is *discretionary*; at the time a word is formatted into nodes, GNU `troff` doesn't know where the output line will break. Later, when processing a pending output line, GNU `troff` has that knowledge, and iterates through the output line's node list, exercising its discretion to discard these hyphen glyph nodes everywhere except when hyphenating a word at the end of the line.

GNU troff inserts elements 1, 7, and 8; the latter two (which are always present) specify the vertical extent of the last line, possibly modified by \x. The br request finishes the pending output line, inserting a newline token, which is subsequently converted to a space when the diversion is interpolated. Note that the word space node has a fixed width that isn't adjustable anymore. To convert horizontal space nodes back into 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.

The chop request simply reduces the number of elements in a macro, string, or diversion by one. Exceptions are *compatibility save* and *compatibility ignore* tokens, which are ignored. The substring request also ignores those tokens.

Some requests like tr or cflags work on glyph identifiers only; this means that the associated glyph can be changed without destroying this association. This can be very helpful for substituting glyphs. In the following example, we assume that glyph 'foo' isn't available by default, so we provide a substitution using the fchar request and map it to input character 'x'.

```
.fchar \[foo] foo
.tr x \[foo]
```

Now let us assume that we install an additional special font 'bar' that has glyph 'foo'.

```
.special bar
.rchar \[foo]
```

Since glyphs defined with fchar are searched before glyphs in special fonts, we must call rchar to remove the definition of the fallback glyph. Anyway, the translation is still active; 'x' now maps to the real glyph 'foo'.

Macro and request arguments preserve compatibility mode enablement.

```
.cp 1     \" switch to compatibility mode
.de xx
\\$1
..
.cp 0     \" switch compatibility mode off
.xx caf\['e]
    ⇒ café
```

Since compatibility mode is enabled while de is invoked, the macro xx enables compatibility mode when it is called. Argument $1 can still be handled properly because it inherits the compatibility mode enablement status that was active at the point where xx was called.

After interpolation of the parameters, the compatibility save and restore tokens are removed.

5.38 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

The `roff` language family is not the easiest to debug, in part thanks to its design features of recursive interpolation and the use of multi-stage pipeline processing in the surrounding system. Nevertheless there exist several features useful for troubleshooting.

Preprocessors use the `lf` request to preserve the identity of the line numbers and names of input files. GNU `troff` emits a variety of error diagnostics and supports several categories of warning; each can be independently issued or suppressed. A trace of the formatter's input processing stack can be emitted when errors or warnings occur by means of GNU `troff`'s `-b` option, or produced on demand with the `backtrace` request. 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. Examine the state of the formatter with requests that write lists of defined names (macros, strings, and diversions), colors, composite character mappings, environments, occupied font mounting positions, font translations, automatic hyphenation codes and exception words, registers, open streams, and page location traps. Requests can also disclose to the standard error stream the internal properties and representations of characters and classes, macros (and strings and diversions), and the list of output nodes corresponding to the pending output line. Recall Section 5.37 [GNU `troff` Internals], page 237.

`.lf` *input-line-number* [["]*file-identifier*] [Request]

Set the input line number (and, optionally, the file name) the formatter uses when reporting diagnostics. The argument becomes the input line number of the *next* line the formatter reads. *file-identifier* is a sequence of ordinary characters and/or spaces. GNU `troff` removes a leading neutral double quote '`"`' from *file-identifier*, permitting initial embedded spaces in it, and reads it to the end of the input line in copy mode. Recall Section 5.24.2 [Copy Mode], page 188.

`lf`'s primary purpose is to aid the debugging of documents that undergo preprocessing. Programs like `tbl` that transform input in their own languages into `roff` requests use it so that any diagnostic messages emitted by a subsequent preprocessor or by `troff` correspond to the source document.

`.tm` *terminal-message* [Request]
`.tm1` ["]*message* [Request]
`.tmc` ["]*message* [Request]

Send *terminal-message* to the standard error stream. The formatter reads the argument to the end of the input line in copy mode (recall

Section 5.24.2 [Copy Mode], page 188), but does *not* remove a leading double quote; contrast `tm1`.

`tm1` removes a leading neutral double quote '`"`' from *message*, permitting initial embedded spaces in it.

`tmc` works as `tm1`, but does not append a newline.

`.ab` [*terminal-message*] [Request]
Write any *terminal-message* to the standard error stream (like `tm`) and then abort the formatter; that is, stop processing and terminate with a failure status. Aborting does not flush a partially collected line, a potentially significant fact if you're using `ab` to "bisect" a troublesome document or macro definition; see the `fl` request below.

`.ex` [Request]
Exit the formatter; that is, stop processing and terminate successfully. To stop processing only the current file, use the `nx` request; recall Section 5.34 [Host System Service Access], page 228.

When doing something complicated, 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 you know in advance that there are many errors and no useful output, or are interested *only* in diagnostic output, you can suppress GNU `troff`'s formatted output with its `-z` option.

`.pchar` *c* ... [Request]
Report, to the standard error stream, information about each character (be it ordinary, special, or indexed) or character class *c*. Spaces need not separate the arguments. A character defined by a request (`char`, `fchar`, `fschar`, or `schar`) reports its contents as a JSON-encoded string, but the output is not otherwise in JSON format.

`.pcolor` [*col* ...] [Request]
Report, to the standard error stream, each defined color named *col*, its color space identifier, and channel value assignments, or, without arguments, those of all defined colors. A device's default stroke and/or fill colors, "default", are not listed since they are immutable and their details unknown to the formatter.

`.pcomposite` [Request]
Report, to the standard error stream, the list of configured composite character mappings. Recall the `composite` request description in Section 5.19.4 [Characters and Glyphs], page 146. The "from" code point is listed first, followed by its "to" mapping.

`.pev` [Request]
Report the state of the current environment followed by that of all other environments to the standard error stream.

`.pfp` [Request]
Report, to the standard error stream, the list of occupied font mounting positions. Recall the `fp` request description in Section 5.19.1 [Selecting Fonts], page 141. Occupied mounting positions are listed, one per line, in increasing order, followed by the typeface name; if the name corresponds to an abstract style, the entry ends there. Otherwise, the name of the font description file and the font's "internal name" datum, the meaning of which varies by output device, follow.

`.pftr` [Request]
Report, to the standard error stream, the list of font translations. Recall the `ftr` request description in Section 5.19.1 [Selecting Fonts], page 141. The "from" font identifier is listed first, followed by its "to" translation.

`.phw` [*word* ...] [Request]
Report, to the standard error stream, the hyphenation points of each hyphenation exception word *word* defined by the `hw` request, or, without arguments, the list of hyphenation exception words associated with the hyphenation language selected by the `hla` request; recall Section 5.10 [Manipulating Hyphenation], page 114. Hyphens in *word* are ignored. A '`-`' marks each hyphenation point. A word prefixed with '`-`' is not hyphenated at all. The report suffixes words to which automatic hyphenation applies (meaning those defined in a hyphenation pattern file rather than with the `hw` request) with a tab and asterisk (`*`).

`.pline` [Request]
Report, in JSON syntax to the standard error stream, the list of output nodes corresponding to the pending output line. In JSON, a pair of empty brackets '`[ ]`' represents an empty list. A *pending* output line has not yet undergone adjustment, and lacks a line number and margin character (all as applicable).

`.pm` [*name* ...] [Request]
Report, to the standard error stream, the JSON-encoded name and contents of each macro, string, or diversion *name*, or, without arguments, the names of all defined macros, strings, and diversions and their lengths in characters or nodes.

`.pnr` [*reg* ...] [Request]
Report the name, value, and (as applicable) autoincrement amount and assigned format of each register *reg* to the standard error stream. If arguments are omitted, report the foregoing properties of all defined registers.

`.pstream` [Request]

Report, in JSON syntax to the standard error stream, the list of open streams, including the name of each open stream, the name of the file backing it, and its mode (writing or appending). In JSON, a pair of empty brackets '`[ ]`' represents an empty list.

`.pwh` [Request]

If the top-level diversion is the current diversion, report the names and positions of all page location traps to the standard error stream. GNU `troff` reports empty slots in the list, where a trap had been planted but subsequently (re)moved, because they can affect the visibility of subsequently planted traps. Otherwise, similarly report the name and position of the diversion trap, if any.

`.fl` [Request]

Break the line and flush any pending output line immediately. The effect is the same as the `br` request unless the no-break control character is used; '`'br`' does nothing, whereas '`'fl`' writes the pending output line without further updating the drawing position. However, the *reported* horizontal drawing position is still reckoned from the start of the input line.

```
foo \n(hp
bar \c
'fl
\n(hp baz \n(hp
    ⇒ foo 96 bar 0 baz 144
```

Flush timing is most easily perceived in device-independent output.

Use of '`'fl`' may be desirable immediately prior to an `ab` request when troubleshooting a document or macro definition line by line, because a significant number of formatting operations can accumulate on a partially collected output line, misleading you about "where" the abort "really" took place.

Historically, `fl` was used with `rd` to produce interactive `nroff` documents. GNU `troff` does not easily support that mode of operation, because its output for terminals is first prepared in device-independent format, which `grotty` renders a page at a time.

`.backtrace` [Request]

Write the state of the input stack to the standard error stream.

Consider the following in a file `test`.

```
.de xxx
.  backtrace
..
.de yyy
.  xxx
..
.
.yyy
     error  troff: backtrace: 'test':2: macro 'xxx'
     error  troff: backtrace: 'test':5: macro 'yyy'
     error  troff: backtrace: file 'test':8
```

The `-b` option of GNU `troff` causes a backtrace to be generated on each error or warning. Some warnings have to be enabled; see Section 5.38.1 [Warnings], page 245.

`\n[slimit]` [Register]

If greater than 0, sets the maximum quantity of objects on GNU `troff`'s internal input stack. If less than or equal to 0, there is no limit: recursion can continue until program memory is exhausted. The default is 1,000.

`.warnscale` *scaling-unit* [Request]

Set the scaling unit used in certain warnings (one of '`u`', '`i`', '`c`', '`p`', and '`P`'; default: '`i`'). Ignored on `nroff`-mode output devices, for which these diagnostics report the vertical page location in lines, and the horizontal page location in ens.

`.spreadwarn` [*limit*] [Request]

Emit a `break` warning if the additional space inserted for each space between words in an output line adjusted to both margins with '`.ad b`' is larger than or equal to *limit*. A negative value is treated as zero; an absent argument toggles the warning on and off without changing *limit*. The default scaling unit 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 GNU `troff` must add 0.2 m or more for each inter-word space in a line. See Section 5.38.1 [Warnings], page 245.

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

Select categories of warnings to be reported. *n* is the sum of the numeric codes associated with each warning category that is to be enabled; all other categories are disabled. The categories and their associated codes are listed in Section 5.38.1 [Warnings], page 245. For example, '`.warn 0`' disables all warnings, and '`.warn 1`' disables all warnings except those

about missing glyphs. Omitting the argument enables all warning categories.

The read-only register .warn contains the sum of the numeric codes of enabled warning categories.

GNU troff has command-line options for reporting warnings (-w), suppressing them (-W), and issuing backtraces (-b) when a warning or an error occurs.

5.38.1 Warnings

GNU troff divides its warning diagnostics into named, numbered categories. The -w and -W options use the associated names. A power of two encodes each category; the warn request and the .warn register respectively set and report the sum of enabled category codes. Warnings of each category are produced under the following circumstances.

'char'
'1' No user-defined character of the requested name or index exists and no mounted font defines a glyph for it, input could not be encoded for device-independent output, or a character without a hyphenation code appeared in a hyphenation exception word. This category is enabled by default.

'break'
'4' A filled output line could not be broken such that its length was less than or equal to, or adjusted such that its length was exactly equal to, the output line length '\n[.l]'. GNU troff reports the amount of overset or underset in the scaling unit configured by the warnscale request in troff mode, and in ens ('n'; character cells) in nroff mode. Recall Section 5.14 [troff and nroff Modes], page 130. This category is enabled by default.

'delim'
'8' The selected delimiter character was ambiguous because it is also meaningful when beginning a numeric expression, or the closing delimiter in an escape sequence was missing or mismatched. This category is enabled by default.

A future groff release may reject ambiguous delimiters. In compatibility mode, ambiguous delimiters are accepted without warning.

'style'
'16' Input was non-idiomatic or likely to produce an unexpected result (or none at all), but is not invalid: a brace escape sequence was employed uselessly; the end of a sentence was detected prior to the end of a text line; or an attempt was made to combine temporary indentation with centering or right-alignment.

‘scale’
‘32’ A scaling unit inappropriate to its context was used in a numeric expression.

‘range’
‘64’ A numeric expression was out of range for its context.

‘syntax’
‘128’ A self-contradictory hyphenation mode or character flags were specified; an empty or incomplete numeric expression was encountered; an operand to a numeric operator was missing; an attempt was made to format characters or spaces on an input line after an output line continuation escape sequence; a recognized but inapposite escape sequence or unprintable character was employed in a device extension command; an attempt was made to define a recursive, empty, or nonsensical character class; or a `groff` extension escape sequence or conditional expression operator was used while in compatibility mode. This category is enabled by default.

‘di’
‘256’ A `di`, `da`, `box`, or `boxa` request was invoked without an argument when there was no current diversion.

‘mac’
‘512’ An undefined string, macro, or diversion was used. When such an object is dereferenced, an empty one of that name is automatically created. So, unless it is later deleted, GNU `troff` issues at most one warning for each.

GNU `troff` also uses this category to warn of an attempt to move an unplanted trap macro; recall Section 5.29.1.1 [Page Location Traps], page 206. In such cases, the unplanted macro is *not* dereferenced, so it is not created if it does not exist.

‘reg’
‘1024’ An undefined register was used. When an undefined register is dereferenced, the formatter automatically defines it with a value of 0. So, unless it is later deleted, GNU `troff` issues at most one warning for each.

‘tab’
‘2048’ A tab character appeared in a parameterized escape sequence, in an unquoted macro argument, or where a request expected a numeric expression argument.

‘missing’
‘8192’ A request was invoked with a mandatory argument absent.

'input'
'16384' An invalid character occurred on the input stream, or was present in the contents of a file processed as transparent throughput.

'escape'
'32768' An unsupported escape sequence was encountered. This category is enabled by default.

'space'
'65536' A space was missing between a request or macro and its argument. This warning is produced when an undefined name longer than two characters is encountered and the first two characters of the name constitute a defined name. No request is invoked, no macro called, and an empty macro is not defined. This category is enabled by default. It never occurs in compatibility mode.

'font'
'131072' An attempt was made to select a font that does not exist. This category is enabled by default.

'ig'
'262144' An invalid escape sequence occurred in input ignored using the `ig` request. This warning category diagnoses a condition that is an error when it occurs in non-ignored input.

'color'
'524288' An attempt was made to select a nonexistent color name, to define a color using an unrecognized color space, or to redefine a default color; or an invalid channel value in a color definition was encountered.

'file'
'1048576' An attempt was made to read a file that does not exist, or a stream remained open at formatter exit. This category is enabled by default.

Two warning names group other warning categories for convenience.

'all' All warning categories except '`di`', '`mac`', '`reg`', and '`style`'. This shorthand is intended to produce all warnings that are useful with macro packages written for AT&T `troff` and its descendants, which have less fastidious diagnostics than GNU `troff`.

'w' All warning categories. Authors of documents and macro packages targeting `groff` are encouraged to use this setting.

5.39 Implementation Differences

GNU `troff` has a number of features that cause incompatibilities with documents written for other versions of `troff`. Some GNU extensions to `troff` have become supported by other implementations.

5.39.1 Safer Mode

GNU `troff` operates in *safer mode* by default; to mitigate risks from untrusted input documents, it disables the `cf`, `pi`, and `sy` requests. GNU `troff`'s `-U` option enables "unsafe mode", restoring their function and enabling additional extension requests, `open`, `opena`, and `pso`. Recall Section 5.34 [Host System Service Access], page 228.

5.39.2 Compatibility Mode

GNU `troff`'s features sometimes cause incompatibilities with documents written assuming old implementations of `troff`. Some syntactical and behavioral differences between them are thought too important to neglect; GNU `troff` therefore makes available a *compatibility mode* in an effort to keep documents prepared for AT&T `troff` rendering well.

Identifiers of arbitrary length 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 input as calling a macro named `dsabcd`. AT&T `troff` also interprets '`\*[`' and '`\n[`' as interpolating a string or register, respectively, named '`[`'. GNU `troff`, however, normally interprets '`[`' as bracketing a long name (with '`]`' at the distal end). In compatibility mode, GNU `troff` interprets names in the traditional way; they thus can be two characters long at most.

`.cp` [*b*] [Request]
`\n[.C]` [Register]

Enable or disable AT&T `troff` compatibility mode per Boolean expression *b*. It is disabled by default, and enabled if *b* is omitted. In compatibility mode, long names are not recognized, and the incompatibilities they cause do not arise.

The read-only register `.C` interpolates 1 if compatibility mode is enabled, 0 otherwise.

GNU `troff`'s `-C` command-line option causes it to start up in compatibility mode.

`.do` *name* [*argument* ...] [Request]
`\n[.cp]` [Register]

Interpret 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 interpolation of *name* is interpreted; that is, the restored compatibility state applies to the request or contents of the macro, string, or diversion *name*, its arguments, and

data read from files or pipes if *name* is the so, soquiet, mso, msoquiet, 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, interpolates 1 if compatibility mode was on when the do request was encountered, and 0 if it was not. This register is specialized and may require a statement of rationale.

When writing macro packages or documents that use GNU troff features and which may be mixed with other packages or documents that do not—common scenarios include serial processing of man pages or use of the so or mso requests—you may desire correct operation regardless of compatibility mode enablement in the surrounding context. It may occur to you to save the existing value of '\n(.C' into a register, say, '_C', at the beginning of your file, turn compatibility mode off with '.cp 0', then restore it from that register at the end with '.cp \n(_C'. At the same time, a modular design of a document or macro package may lead you to multiple layers of inclusion. You cannot use the same register name everywhere lest you "clobber" the value from a preceding or enclosing context. The two-character register name space of AT&T troff is confining, but employing GNU troff's more capacious one, as with '.nr _my_saved_C \n(.C', does not work in compatibility mode; the register name is too long. Employing the do request is no help: '.do nr _my_saved_C \n(.C' always saves zero to the register, because do turns compatibility mode *off* while it interprets its argument list.

To robustly save compatibility mode before switching it off, use

```
.do nr _my_saved_C \n[.cp]
.cp 0
```

at the beginning of your file, followed by

```
.cp \n[_my_saved_C]
.do rr _my_saved_C
```

at the end. As the C language exposes application programs' symbols to those defined by libraries, `roff` documents share a name space with macro packages; choose a register name that is unlikely to collide with other uses.

Normally, GNU `troff` tracks the nesting depth of interpolations. In compatibility mode, it does not.

```
.ds xx '
\w'abc\*(xxdef'
```

⇒ `168` (not in compatibility mode on a terminal device)
⇒ `72def'` (compatibility mode on a terminal device)

In AT&T `troff`, the escape sequences `\f`, `\H`, `\m`, `\M`, `\R`, `\s`, and `\S` are transparent to control character recognition at the beginning of a line, or after the conditional expression of an `if` or `ie` request. After reading an escape sequence that starts an input line, GNU `troff` normally no longer recognizes a control character on the input line; but in compatibility mode, it does. Thus the next example produces bold output in both modes, but the text differs.

```
.de xx
Hello!
..
\fB.xx\fP
```

⇒ `.xx` (not in compatibility mode)
⇒ `Hello!` (in compatibility mode)

Normally, the syntax form `\s`*n* accepts only a single character (a digit) for *n*, consistently with other forms that originated in AT&T `troff`, like `\*`, `\f`, `\g`, `\k`, `\n`, and `\z`. In compatibility mode only, a non-zero *n* must be in the range 4–39. Legacy documents relying upon this quirk of parsing[284] should migrate to another `\s` form.

In compatibility mode, the `de`, `am`, `ds`, and `as` requests behave as `de1`, `am1`, `ds1`, and `as1`, respectively: GNU `troff` inserts a *compatibility save* token at the beginning of the macro, string, or appendment thereto as applicable and a *compatibility restore* token at its end, enabling compatibility mode during

[284] The Graphic Systems C/A/T phototypesetter (the original device target for AT&T `troff`) supported only a few discrete type sizes in the range 6–36 points, so Ossanna contrived a special case in the parser to do what the user must have meant. Kernighan warned of this in the 1992 revision of CSTR #54 (§2.3), and more recently, McIlroy referred to it as a "living fossil".

its interpolation; recall Section 5.22 [Strings], page 170. Thus they work as expected even if the interpolation context disables compatibility mode.

AT&T troff recognized slightly varying sets of delimiters when expecting numerical expressions (as with the \h escape sequence), string expressions (as with the \w escape sequence and tl request), and output comparisons (as in '.if #foo#bar# .tm match'). GNU troff, when not in compatibility mode, recognizes a single consistent set of delimiters. Compatibility mode emulates AT&T troff only up to a point. GNU troff accepts leaders and tabs as delimiters, as well as Control+D (EOT or EOF), Control+H (BS or backspace), and Control+L (FF or form feed), all of which, when used as delimiters, cause AT&T troff to behave in ways difficult to predict.

5.39.3 Other Differences

In GNU troff, read-only registers are subject to removal like others; once removed, they can be recreated only as normal writable registers and will not otherwise reflect the configuration of the formatter.

Solaris and Heirloom Doctools troffs' rm and rr requests halt argument processing upon encountering an invalid identifier. Those of GNU, DWB, and Plan 9 from User Space troffs do not; they remove objects named by any subsequent valid arguments.

GNU troff does not emit output if it has nothing to format. For example, it treats an input document consisting solely of nr and tm requests as empty, and produces nothing on its standard output stream. AT&T troff does, creating a blank page.

Use of C0 control characters in identifiers is not portable; Solaris, Plan 9, and Heirloom Doctools troffs accept Control+B, Control+C, Control+E, Control+F, and Control+G (only); DWB 3.3 troff does not. GNU troff rejects C0 controls in identifiers with an error diagnostic.

In DWB and Plan 9 from User Space troff, spaces or tabs between the ordinary control character and a '.' or macro name that would normally end a macro definition prevent it from doing so.

GNU troff regards a failed attempt to select a different font as a meaningful action and updates the the identity of the previous font selection. So does Heirloom Doctools troff. Other AT&T-descended troffs, and mandoc, do not. Consequently, given the input

```
\fBfoo\fZbar\fPbaz
```

and, assuming that the output device lacks a font named 'Z', GNU and Heirloom Doctools troffs set 'baz' in bold; DWB, Plan 9, and Solaris troffs set it in roman. The only way to determine if a font selection operation succeeded in the latter troffs is to save the value of the .f register prior to attempting the change, make the attempt, then compare the saved value to the current one; if they differ, the font selection presumably succeeded.[285]

[285] *mandoc*(1) does not implement a .f register.

Since doing so requires `nr` and `if` (or `ie`) requests, such a procedure cannot be performed amid `\f` sequences on a text line.

AT&T `troff`'s `yr` register has a Y2K bug; GNU `troff` simulates the bug for consistency due to its widespread usage, but provides an alternative register implementing AT&T's *documented* semantics; recall Section 5.34 [Host System Service Access], page 228.

Formatters that don't implement GNU `troff` extension request names tend to ignore them.

If a formatter doesn't support a GNU `troff` extension escape sequence, it is liable to format its function selector character as text. For example, the adjustable, non-breaking space escape sequence `\~` is also supported by Heirloom Doctools `troff` 050915 (September 2005), `mandoc` 1.9.5 (2009-09-21), `neatroff` (commit 1c6ab0f6e, 2016-09-13), and Plan 9 from User Space `troff` (commit 93f8143600, 2022-08-12), but not by Solaris or Documenter's Workbench `troff`s, which both render it as '`~`'. Recall Section 5.9 [Manipulating Filling and Adjustment], page 106.

AT&T `troff` discards trailing spaces from input lines, like GNU `troff`, but when it does so, AT&T `troff` also cancels end-of-sentence detection. Use of the dummy character escape sequence `\&` is more portable.

When adjusting output lines to both margins, AT&T `troff` at first adjusts spaces starting from the right; GNU `troff` begins from the left. Both implementations adjust spaces from opposite ends on alternating output lines in this adjustment mode to prevent "rivers" in the text.

GNU `troff` does not always hyphenate words as AT&T `troff` does. The AT&T implementation uses a set of hard-coded rules specific to U.S. English ("suftab"), while GNU `troff` uses language-specific hyphenation pattern files derived from TEX. Some versions of `troff` reserved meager storage for hyphenation exception words (arguments to the `hw` request); GNU `troff` has no such restriction. When the `hy` request is invoked without an argument, GNU `troff` sets the automatic hyphenation mode to the value of the `.hydefault` register; AT&T `troff` sets it to '`1`', which is not suitable in GNU `troff` for some languages, including English. Recall the table in Section 5.10 [Manipulating Hyphenation], page 114.

GNU `troff` handles the dummy character `\&` differently from AT&T `troff` when it is followed by the hyphenation control escape sequence `\%` at the beginning of a word. GNU `troff` does not regard the dummy character as "starting" the word; AT&T `troff` does. Further, Heirloom Doctools `troff` does not honor an explicit hyphenation point marked with `\%` after a word-initial one.[286]

GNU `troff` interprets a request argument representing a file name or system command in the same way it does the *contents* argument to the `ds` and `as` requests: it removes a leading neutral double quote '`"`' from the argument to the `cf`, `nx`, `pi`, `so`, and `sy` requests, and the second argument

[286] Thus,

(if present) to the lf request, permitting initial embedded spaces in it, and reads it to the end of the input line in copy mode. Recall Section 5.24.2 [Copy Mode], page 188. This difference permits GNU troff to handle files with spaces in their names, but requires more care with trailing comments, and doubling of an initial neutral double quote '"' if the file name has one.

GNU troff accepts multiple leading complementation operators ! in a conditional expression, inverting its truth value with each occurrence.

```
.ie !!0 .tm true
.el     .tm false
    error  false
```

AT&T troff regards the foregoing conditional expression as malformed (but issues no diagnostic message) and takes *neither* branch.

The existence of the .T string is a common feature of device-independent troffs—DWB 3.3, Solaris, Heirloom Doctools, and Plan 9 troffs all support it—but valid values are specific to each implementation.

The (read-only) register .T interpolates 1 if GNU troff is run with the -T option, and 0 otherwise. In contrast, AT&T troff interpolated 1 only if nroff was the formatter and was run with -T.

AT&T troff had only environments named '0', '1', and '2'. In GNU troff, any number of environments may exist, using any valid identifiers for their names. Recall Section 5.5 [Identifiers], page 86.

As noted in Section 5.20.3 [Using Fractional Type Sizes], page 166, AT&T troff's ps request ignores scaling units and thus '.ps 10u' sets the type size to 10 points, whereas in GNU troff it sets the type size to 10 *scaled* points, possibly a much smaller measurement. AT&T's behavior also means that '.ps 10p' and '.ps 10z' are portable.

AT&T troffs did not require a space between the arguments of the dt and wh requests. We know of no legacy documents that require such syntactical flexibility; GNU troff therefore does not support it. It may have been unknown because the numeric expression accepted by the requests could be of arbitrary length.

The ab request differs from that of AT&T troff: GNU troff writes no message to the standard error stream if no arguments are given, and it exits with a failure status instead of a successful one.

The bp request differs from that of AT&T troff: GNU troff does not accept a scaling unit on the argument, a page number; the former does (uselessly).

```
.ll 10n
\%antidisestablishmen\%tarianism
.br
\&\%antidisestablishmen\%tarianism
.pl \n(nlu
```

produces different results with each of the three formatters.

When the `dt` request is given invalid arguments, AT&T `troff` behaves as if it were given no arguments at all, removing any existing diversion trap; GNU `troff` does nothing (besides issue diagnostics), more consistently with other requests given unintelligible arguments.

The `lf` request sets the number of the *current* input line in AT&T `troff` and the *next* in GNU `troff`.

In AT&T `troff`, the `pm` request reports macro, string, and diversion sizes in units of 128-byte blocks, and an argument reduces the report to a sum of the above in the same units. GNU `troff` reports their lengths in characters or nodes if given no arguments, and otherwise dumps the JSON-encoded name, contents, and other properties of each named argument.

The `sp` request differs from that of AT&T `troff`: invoking `sp` with the no-break control character while "not on a page"—that is, when writing output to the top-level diversion while the vertical drawing position reported by register `nl` is negative—provokes an error diagnostic and does not move the drawing position. AT&T `troff` sets the vertical drawing position to two basic units fewer than the specified argument, and to one vee minus one basic unit if no argument is present.

AT&T `troff` ignores the `ss` request if the output is a terminal device; GNU `troff` rounds down the values of minimum inter-word and additional inter-sentence space each to the nearest multiple of 12.

In AT&T `troff`, the `sv` request springs a page location trap if it is nearer to the drawing position than the amount of vertical space requested; GNU `troff` saves the space but does not spring the trap. To simulate AT&T behavior, consider the following.

```
.sv 3i \" or however much space you need
.if \n(.g .ne 3i
```

GNU `troff` distinguishes characters from glyphs. Characters can be ordinary, special, or indexed, and populate strings and macros. Characters *per se* have not (yet) been formatted. Glyphs represent graphemes (supplied by the output device) and populate diversions (recall Section 5.30 [Diversions], page 217). There is no such thing as a space character in GNU `troff` output. Formatting converts characters into (sequences of) glyphs. GNU `troff` stores properties of the environment that affect how a glyph is rendered with the glyph node's data. Thus, subsequent formatting operations do not affect it, including `bd`, `cs`, `tkf`, `tr`, and `fp` requests. Normally, a macro or string contains only a list of characters and a diversion contains only a list of nodes. However, applying the `asciify` or `unformat` requests to a diversion converts some of its nodes back into characters. Where the formatter cannot recover the character representation of a node, it stores a null character in the character list corresponding to a single node in the node list.

Consequently, a glyph node does not behave as a character does in macro interpolation: it does not inherit special properties that the character from which it was constructed might have had. For example, the input

```
.di x
\\\\
.br
.di
.x
```

produces '\\' in GNU troff. Each pair of backslashes becomes one backslash *glyph*; the resulting backslashes are thus not interpreted as escape *characters* when they are interpolated as the diversion is output. AT&T troff *would* interpret them as escape characters when interpolating them and 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,[287] regardless of whether 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 roff escape character—perhaps in discussion of character sets or other programming languages—is the special character escape sequence \(rs or \[rs], for "reverse solidus", from its name in the ECMA-6 and ISO 10646 standards.[288]

To store in a diversion an escape sequence that is interpreted when the diversion is interpolated, either use the traditional \! transparent output facility, or, if this is unsuitable, the new \? escape sequence. Recall Section 5.30 [Diversions], page 217, and Section 5.37 [GNU troff Internals], page 237.

Like AT&T troff, GNU troff maintains a buffer of device-independent output commands,[289] populating the buffer as formatted output accumulates. GNU troff always flushes this buffer when processing a break; AT&T troff does so according to no obvious schedule. Perhaps, if the buffer is of fixed size, the formatter performs the flush when the buffer runs out of room.

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 outputs a partially collected but otherwise empty line (as if '\c' were in the top-level diversion) at the end of input; GNU troff does not.

---

[287] 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.

[288] AT&T troff's font description files did not define the rs special character, but those of its descendant Heirloom Doctools troff do, as of its 060716 release (July 2006).

[289] In GNU troff, node objects produce these commands; recall Section 5.37 [GNU troff Internals], page 237.

6 File Formats

All files that GNU `troff` reads and writes are text files.[1] The next two sections describe their format.

6.1 Device and Font Description Files

The `groff` font and output device description formats are slight extensions of those used by AT&T device-independent `troff`. In distinction to the AT&T implementation, `groff` lacks a binary format; all files are text files.[2] The device and font description files for a device *name* are stored in a `dev`*name* directory. The device description file is called `DESC`, and, for each font supported by the device, a font description file is called *f*, where *f* is usually an abbreviation of a font's name and/or style. For example, the `ps` (PostScript) device has `groff` font description files for Times roman (`TR`) and Zapf Chancery Medium italic (`ZCMI`), among many others, while the `utf8` device (for terminals) has font descriptions for the roman, italic, bold, and bold-italic styles (`R`, `I`, `B`, and `BI`, respectively).

Device and font description files are read both by the formatter, GNU `troff`, and by output drivers. The programs delegate these files' processing to an internal library, `libgroff`, ensuring their consistent interpretation.

All integers used by these files must be specified in decimal format.

6.1.1 `DESC` File Format

The `DESC` file contains a series of directives; each begins a line. Their order is not important, with two exceptions: (1) the `res` directive must precede any `papersize` directive; and (2) the `charset` directive must come last (if at all). If a directive name is repeated, later entries in the file override previous ones (except that the paper dimensions are computed based on the `res` directive last seen when `papersize` is encountered). Spaces and/or tabs separate words and are ignored at line boundaries. Comments start with the '`#`' character and extend to the end of a line. Empty lines are ignored.

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

`fonts` *n F1* ... *Fn*
: Fonts *F1*, ..., *Fn* are mounted at positions *m*+1, ..., *m*+*n*, where *m* is the number of `styles` (see below). This directive may extend over more than one line. A font name of `0` causes no font to be mounted at the corresponding position.

---

[1] GNU `troff` also reads files that don't satisfy the strict POSIX definition of a text file—for example, those lacking a final newline character—and the `cf` and `trf` requests read arbitrary files. Recall Section 5.34 [Host System Service Access], page 228.

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

`hor` *n*
: The horizontal motion quantum is *n* basic units. Horizontal measurements round to multiples of *n*.

`image_generator` *program*
: Use *program* to generate PNG images from PostScript input. Under GNU/Linux, this is usually `gs`, but under other systems (notably Cygwin) it might be set to another name. The `grohtml` driver uses this directive.

`paperlength` *n*
: The vertical dimension of the output medium is *n* basic units (deprecated: use `papersize` instead).

`papersize` *format-or-dimension-pair-or-file-name* ...
: The dimensions of the output medium are as according to the argument, which is either a standard paper format, a pair of dimensions, or the name of a plain text file containing either of the foregoing.

  Recognized paper formats are the ISO and DIN formats `A0`–`A7`, `B0`–`B7`, `C0`–`C7`, `D0`–`D7`; the U.S. paper types `letter`, `legal`, `tabloid`, `ledger`, `statement`, and `executive`; and the envelope formats `com10`, `monarch`, and `DL`. Matching is performed without regard for lettercase.

  Alternatively, the argument can be a custom paper format in the format *length*`,`*width* (with no spaces before or after the comma). Both *length* and *width* must have a unit appended; valid units are '`i`' for inches, '`c`' for centimeters, '`p`' for points, and '`P`' for picas. Example: '`12c,235p`'. An argument that starts with a digit is always treated as a custom paper format.

  Finally, the argument can be a file name (e.g., `/etc/papersize`); if the file can be opened, the first line is read and a match attempted against each of the other forms. No comment syntax is supported.

  More than one argument can be specified; each is scanned in turn and the first valid paper specification used.

`paperwidth` *n*
: The horizontal dimension of the output medium is *n* basic units (deprecated: use `papersize` instead).

`pass_filenames`
: Direct GNU `troff` to emit the name of the source file being processed. This is achieved with the intermediate output command '`x F`', which `grohtml` interprets.

`postpro` *program*
: Use *program* as the postprocessor.

`prepro` *`program`*
: Use *program* as a preprocessor. The `html` and `xhtml` output devices use this directive.

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

`res` *`n`*
: The device resolution is *n* basic units per inch.

`sizes` *`s1 ... sn`* `0`
: The device has fonts at *s1*, . . . , *sn* scaled points (see below). The list of sizes must be terminated by `0`. Each *si* can also be a range of sizes *m*–*n*. The list can extend over more than one line.

`sizescale` *`n`*
: A typographical point is subdivided into *n* scaled points. The default is `1`. Recall Section 5.20.3 [Using Fractional Type Sizes], page 166.

`styles` *`S1 ... Sm`*
: The first *m* mounting positions are associated with styles *S1*, . . . , *Sm*.

`tcommand`
: The postprocessor interprets the `t` and `u` output commands that GNU `troff` produces.

`unicode`
: The output device supports the complete Unicode repertoire. This directive is useful only for devices that produce character entities instead of glyphs.

  If `unicode` is present, no `charset` section is required in the font description files since the Unicode handling built into `groff` is used. However, if there are entries in a font description file's `charset` section, they either override the default mappings for those particular characters or add new mappings (normally for composite characters).

  The `utf8`, `html`, and `xhtml` output devices use this directive.

`unitwidth` *`n`*
: Arbitrary basis with respect to which font metrics are proportionally scaled when rendering glyphs at a type size of one point.

`unscaled_charwidths`
: Make the font handling module always return unscaled character widths. The `grohtml` driver uses this directive.

`use_charnames_in_special`
: GNU `troff` should encode special characters in arguments to device extension commands; recall Section 5.35 [Postprocessor Access], page 235. *grohtml*(1) uses this directive.

`vert` *n* The vertical motion quantum is *n* basic units. Vertical measurements round to multiples of *n*.

`charset` This line and everything following it in the file are ignored. It is recognized for compatibility with other `troff` implementations. In GNU `troff`, character set repertoire is described on a per-font basis.

GNU `troff` recognizes but ignores the directives `spare1`, `spare2`, and `biggestfont`.

The `res`, `unitwidth`, `fonts`, and `sizes` lines are mandatory. Directives not listed above are ignored by GNU `troff` but may be used by postprocessors to obtain further information about the device.

6.1.2 Font Description File Format

On typesetting output devices, each font is typically available at multiple sizes. While paper measurements in the device description file are in absolute units, measurements applicable to fonts must be proportional to the type size. The font's *unit width* establishes a numerical basis that permits all of its metrics to be expressed as integers if rendered at one point. When the formatter configures a type size, it scales the metrics linearly relative to that basis. The unit width has no inherent relationship to the device resolution, and the same division procedure applies to all font metrics. Observe that whatever unit might one select for the unit width, the division operation implied by scaling cancels it out, leaving a dimensionless quantity.

For instance, `groff`'s `lbp` device uses a `unitwidth` directive with an argument of 800. Its Times roman font '`TR`' has a `spacewidth` of 833; this is also the width of its comma, period, centered period, and mathematical asterisk, while its '`M`' has a width of 2,963. Thus, an '`M`' on the `lbp` device is 2,963 ÷ 800 times the unit width, or approximately 3.7. At a type size of 10 points, a Times roman '`M`' is therefore 37 units wide.

```
$ groff -T lbp
.ps 10
.nr Mw \w'M'
.tm width of 'M' at 10 points=\n(Mw
     error  width of 'M' at 10 points=37
```

A font description file has two sections. The first is a sequence of directives, and is parsed similarly to the `DESC` file described above. Except for the directive names that begin the second section, their ordering is immaterial. Later directives of the same name override earlier ones, spaces and tabs are handled in the same way, and the same comment syntax is supported. Empty lines are ignored throughout.

`name` *f*
: The name of the font is *f*. '`DESC`' is an invalid font name. Simple integers are valid, but their use is discouraged.[3]

`spacewidth` *n*
: The width of an unadjusted inter-word space is *n*, relative to the device's unit width.

The directives above must appear in the first section; those below are optional.

`slant` *n*
: The font's glyphs have a slant of *n* degrees; a positive *n* slants in the direction of text flow.

`ligatures` *lig1* ... *lign* [`0`]
: Glyphs *lig1*, ..., *lign* are ligatures; possible ligatures are '`ff`', '`fi`', '`fl`', '`ffi`' and '`ffl`'. For compatibility with other `troff` implementations, the list of ligatures may be terminated with a `0`. The list of ligatures must not extend over more than one line.

`special`
: The font is *special*: when the document attempts to format a glyph that is not present in the formatter's currently selected font, the glyph is sought in any mounted fonts that bear this property. Often, such fonts are *unstyled*, having no heavy (bold) or slanted (italic or oblique) variants.

Other directives in this section are ignored by GNU `troff`, but may be used by postprocessors to obtain further information about the font.

The second section contains one to three subsections, which can appear in any order, and any of which starts the second section. Each starts with a directive on a line by itself. A `charset` subsection is mandatory unless the associated `DESC` file contains the `unicode` directive. Another subsection, `kernpairs`, is optional.

The directive `charset` starts the character set subsection.[4] It precedes a series of glyph descriptions, one per line. Each such glyph description comprises a set of fields separated by spaces or tabs and organized as follows.

name metrics type index [*entity-name*] [`--` *comment*]

name identifies the glyph: if *name* is a printable character *c*, it corresponds to the `troff` ordinary character *c*. If *name* is a multi-character sequence not beginning with `\`, it corresponds to the GNU `troff` special character escape sequence '`\[`*name*`]`'. A name consisting of three minus signs, '`---`', is special and indicates that the glyph is unnamed: such glyphs can be accessed only by the `\N` escape sequence in `troff`. A special character named '`---`' can still be defined using `char` and similar requests. The *name* '`\-`' defines the minus

[3] `groff` requests and escape sequences interpret non-negative integers as mounting positions instead. Further, a font named '`0`' cannot be automatically mounted by the `fonts` directive of a `DESC` file.

[4] On typesetters, this directive is misnamed since it starts a list of glyphs, not characters.

sign glyph. Finally, *name* can be the unbreakable one-sixth and one-twelfth space escape sequences, `\|` and `\^` ("thin" and "hair" spaces, respectively), in which case only the width metric described below is interpreted; a font can thus customize the widths of these spaces.

The form of the *metrics* field is as follows.

width[,[*height*[,[*depth*[,[*italic-correction*
[,[*left-italic-correction*[,[*subscript-correction*]]]]]]]]]

Spaces, tabs, and newlines are prohibited between these *subfields*, which are expressed as decimal integers (and have been split here into two lines only for better legibility). The unit of measure is that established by the `unitwidth` directive and scaled to the type size. Unspecified subfields default to `0`. Since there is no associated binary format, these values are not required to fit into the C language data type '`char`' as they are in AT&T device-independent `troff`.

The *width* subfield gives the advance width of the glyph; recall [Artificial Fonts], page 156. The *height* subfield gives the height of the glyph (upward is positive); if a glyph does not extend above the baseline, give it a zero height, not a negative height. The *depth* subfield gives the depth of the glyph—that is, the distance below the baseline to which the glyph extends (downward is positive); if a glyph does not extend below the baseline, give it a zero depth, not a negative depth. Italic corrections apply when upright and slanted (italic or oblique) styles are typeset adjacently. The *italic-correction* is the amount of space to add after a slanted glyph to be followed immediately by an upright glyph. The *left-italic-correction* is the amount of space to add before a slanted glyph to be preceded immediately by an upright glyph. The *subscript-correction* is the amount of space to add after a slanted glyph to be followed by a subscript; it should be less than the italic correction.

For fonts used with typesetters, the *type* field gives a featural description of the glyph: it is a bit pair recording whether the glyph is an ascender, descender, both, or neither. When a `\w` escape sequence is interpolated, these values are bitwise or-ed together for each glyph and stored in the `ct` register. In font descriptions for terminals, all glyphs might have a type of zero, regardless of their appearance.

`0` means the glyph lies entirely between the baseline and a horizontal line at the "x-height" of the font; typical examples are '`a`', '`c`', and '`x`';

`1` means the glyph descends below the baseline, like '`p`';

`2` means the glyph ascends above the font's x-height, like '`A`' or '`b`'; and

`3` means the glyph is both an ascender and a descender—this is true of parentheses in some fonts.

The *index* field is an integer that uniquely identifies a glyph within the font; any integer is accepted as input,[5] but no practical font employs all possible values. An *index* is limited to the range of the system's C language data type `int`. In a `troff` document, use the indexed character escape sequence `\N` to specify a glyph by index.

The *entity-name* field defines an identifier for the glyph that the postprocessor uses to print the GNU `troff` glyph *name*. This field is optional; it was introduced so that the `grohtml` output driver could encode its character set. For example, the glyph '`\[Po]`' is represented by '`£`' in HTML 4.0. For efficiency, these data are now compiled directly into `grohtml`. `grops` uses the field to build sub-encoding arrays for PostScript fonts containing more than 256 glyphs. Anything on the line after the *entity-name* field or '`--`' is ignored.

A line in the `charset` section can also have the form

name "

identifying *name* as another name for the glyph mentioned in the preceding line. Such aliases can be chained.

A `charset-range` subsection works like the `charset` directive except that the glyph descriptions use a *name* of the form `u`*AAAA*`..u`*FFFF*, where *AAAA* and *FFFF* are hexadecimal digit sequences; the specified metrics then apply identically to all glyphs in the designated range.

The directive `kernpairs` starts a list of kerning adjustments to be made to adjacent glyph pairs from this font. It contains a sequence of lines formatted as follows.

g1 g2 n

The foregoing means that when glyph *g1* is typeset immediately before *g2*, the space between them should be increased by *n*. The unit of measure is that established by the `unitwidth` directive and scaled to the type size. Most kerning pairs should have a negative value for *n*.

6.2 GNU `troff` Output

We now describe the `groff` device-independent page description language produced by GNU `troff`.

As `groff` is a wrapper program around GNU `troff` and automatically runs an output driver, users seldom encounter this format under normal circumstances. `groff` offers the option `-Z` to inhibit postprocessing such that GNU `troff`'s output is sent to the standard output stream just as it is when running GNU `troff` directly.

The purpose of device-independent output is to facilitate the development of postprocessors by providing a common programming interface to all devices. It is a distinct, and much simpler, language from that of the

[5] that is, any integer parsable by the C standard library's *strtol*(3) function

formatter, `troff`. The device-independent output can be thought of as a "page description language".

In the following discussion, the term *troff output* describes what is output by GNU `troff`, while *page description* denotes the language accepted by the parser that interprets this output for the output drivers. This parser handles whitespace (spaces, tabs, and newlines) more flexibly than AT&T `troff`'s implementation, recognizes a GNU extension to the language, and supports a legacy compressed encoding of a subset of commands for compatibility; otherwise, the formats are the same.[6]

When Brian Kernighan designed AT&T `troff`'s device-independent page description language circa 1980, he had to balance readability and maintainability against severe constraints on file size and transmission speed to the output device.[7] A decade later, when James Clark wrote `groff`, these constraints were no longer as tight.

6.2.1 Language Concepts

The fundamental operation of the GNU `troff` formatter is the translation of the `groff` input language into a series of instructions concerned primarily with placing glyphs or geometric objects at specific positions on a rectangular page. In the following discussion, the term *command* always refers to this device-independent output language, and never to the language intended for direct use by document authors. Device-independent output commands comprise several categories: glyph output; font, color, and text size selection; motion of the drawing position; page advancement; drawing of geometric objects; and device control commands, a catch-all for other operations. The last includes directives to start and stop output, identify the intended output device, and embed URL hyperlinks in supported output formats.

6.2.1.1 Syntax

`roff`'s page description language is a sequence of *tokens*: single-letter commands or their arguments. Some commands accept a subcommand as a first argument, followed by one or more further arguments.

AT&T device-independent `troff` used whitespace minimally when producing output. GNU `troff`, in contrast, attempts to make its output more human-readable. The whitespace characters—tab, space, and newline—are always meaningful. They are never used to represent spacing in the document; that is done with horizontal (`h`, `H`) and vertical (`v`, `V`) positioning commands. Any sequence of space and/or tab characters is equivalent to a single space, separating commands from arguments and arguments from each other. Space is required only where omitting it would cause ambiguity. A line break separates commands. The comment character is a pound/hash

[6] The parser for device-independent output can be found in the file *`groff-source-dir`*`/src/libs/libdriver/input.cpp`.

[7] See "A Typesetter-independent TROFF", Bell Labs CSTR #97, 1982.

sign (#), and marks the remainder of the line as a comment. A line comprising only whitespace after comment removal does nothing but separate input tokens.

The positioning commands noted above, and the command to write one glyph (c), each take a single argument; the former a signed integer, and the latter a printable ECMA-6/ISO 646:1991 IRV/US-ASCII character. A series of such commands could validly occur without spaces on an input line, but GNU troff follows each with a newline.

Some commands have a more complex syntax; GNU troff extension commands for writing glyph sequences, t and u, accept a variable number of arguments; whitespace must follow them. Those that draw geometric objects (D) or control the device (x) furthermore recognize a variable number of space- or tab-separated words as arguments; such commands thus must end with a newline. In GNU troff, the device extension (sub)command 'x X' uniquely supports a line continuation syntax; a single input line bounds any other.

6.2.1.2 Argument Units

Some commands take integer arguments that are assumed to represent values in a measurement unit, but the letter for the corresponding scaling unit is not written with the output command arguments. Most commands assume the scaling unit '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 whitespace character (space, tab, or newline) always terminates a string argument. A '#' character embedded in a string argument is part of that argument, not the beginning of a comment command. Any character other than a decimal numeral terminates a command that mtakes a numeric argument, and, if not whitespace, is interpreted as the start of a new command.

6.2.1.3 Output Structure

Device-independent troff output is organized into three parts: a header, a body, and a trailer.

The task of the header is to set general device parameters. GNU troff guarantees that its header consists of the following three lines:

```
x T device
x res n h v
x init
```

with the parameters *n*, *h*, and *v* set as outlined in Section 6.2.2.4 [Device Control Commands], page 271. The parser for the device-independent page

description language format is able to interpret additional whitespace and comments as well even in the header.

The body contains the document's visible content. Once an output driver interprets '`x init`', it prepares to handle commands in general. Semantically, the body is page-oriented. The `p` command starts a new page. Positioning, writing, and drawing commands are performed within a page, so they cannot occur before the first `p` command. The output driver reckons absolute positioning (by the `H` and `V` commands) with respect to the current page's origin at the top left corner, and all other positioning relative to the drawing position on the page.

The trailer advances the drawing position to the bottom of the page and informs the device that the document (or "job") has ended. Processing terminates when a '`x stop`' command is encountered; the last line of any GNU `troff` page description output always contains such a command.

6.2.2 Command Reference

This subsection describes all page description output commands, both from AT&T `troff` as well as extension commands issued by GNU `troff`.

6.2.2.1 Comment Command

`#`*anything*<end of line>

Apply comment annotation. Ignore any characters from the '`#`' character up to the next newline.

Each comment can be preceded by arbitrary syntactical space, and every command can be terminated by a comment.

6.2.2.2 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 necessary only when two integer arguments would clash or if the preceding argument ends with a string argument.

`C` *id*<whitespace>

Typeset the glyph of the special character *id*. Trailing syntactical space is necessary to allow special character names of arbitrary length. The drawing position is not advanced.

`c` *g*

Typeset the glyph of the ordinary character *c*. The drawing position is not advanced.

`f` *n*

Select the font mounted at position *n*. *n* cannot be negative.

`H` *n* Horizontally move the drawing position to *n* basic units from the left edge of the page. *n* cannot be negative.

`h` *n* Move the drawing position right *n* basic units. AT&T `troff` allowed negative *n*; GNU `troff` does not produce such values, but `groff`'s output driver library handles them.

`m` *color-scheme* [*component* ...]
Select the stroke color using the *component*s in the color space *scheme*. Each *component* is an integer between 0 and 65535. The quantity of components and their meanings vary with each *scheme*. This command is a `groff` extension.

`mc` *cyan magenta yellow*
Use the CMY color scheme with components cyan, magenta, and yellow.

`md` Use the default color (no components; black in most cases).

`mg` *gray* Use a grayscale color scheme with a component ranging between 0 (black) and 65535 (white).

`mk` *cyan magenta yellow black*
Use the CMYK color scheme with components cyan, magenta, yellow, and black.

`mr` *red green blue*
Use the RGB color scheme with components red, green, and blue.

`N` *n* Typeset the glyph with index *n* in the current font. *n* is normally a non-negative integer. The drawing position is not advanced. The `html` and `xhtml` devices use this command with negative *n* to produce unbreakable space; the absolute value of *n* is taken and interpreted in basic units.

`n` *b* `a` Indicate a break. No action is performed; the command is present to make the output more easily parsed. The integers *b* and *a* describe the vertical space amounts before and after the break, respectively. GNU `troff` issues this command but `groff`'s output driver library ignores it. See `v` and `V` below.

`p` *n* Begin a new page, setting its number to *n*. Each page is independent, even from those using the same number. The vertical drawing position is set to 0. All positioning, writing, and drawing commands are interpreted in the context of a page, so a `p` command must precede them.

`s` *n* Set type size to *n* scaled points (unit `z` in GNU `troff`. AT&T `troff` used unscaled points `p` instead; see Section 6.2.4 [Output Language Compatibility], page 276.

t *xyz*<whitespace>
t *xyz dummy-arg*<whitespace>

Typeset a word *xyz*; that is, set a sequence of ordinary glyphs named *x*, *y*, *z*, . . . , 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). Each glyph is set at the current drawing position, and the position is then advanced horizontally by the glyph's width. A glyph's width is read from its metrics in the font description file, scaled to the current type size, and rounded to a multiple of the horizontal motion quantum. Use the `C` command to emplace glyphs of special characters. The `t` command is a `groff` extension and is output only for devices whose `DESC` file contains the `tcommand` directive; see Section 6.1.1 [DESC File Format], page 257.

u *n xyz*<whitespace>

Typeset word *xyz* with track kerning. As `t`, but after placing each glyph, the drawing position is further advanced horizontally by *n* basic units (`u`). The `u` command is a `groff` extension and is output only for devices whose `DESC` file contains the `tcommand` directive; see Section 6.1.1 [DESC File Format], page 257.

V *n*

Vertically move the drawing position to *n* basic units from the top edge of the page. *n* cannot be negative.

v *n*

Move the drawing position down *n* basic units. AT&T `troff` allowed negative *n*; GNU `troff` does not produce such values, but `groff`'s output driver library handles them.

w

Indicate an inter-word space. No action is performed; the command is present to make the output more easily parsed. Only inter-word spaces on an output line (be they breakable or not) are thus described; those resulting from horizontal motion escape sequences are not. GNU `troff` issues this command but `groff`'s output driver library ignores it. See `h` and `H` above.

6.2.2.3 Graphics Commands

Each graphics or drawing command in the page description language 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.

GNU `troff` output follows AT&T `troff`'s output conventions (no space between command and subcommand, all arguments are preceded by a single space character), but `groff`'s parser allows optional space between the

command letters and makes the space before the first argument optional. As usual, each space can be any sequence of tab and space characters.

Some graphics commands can take a variable number of arguments. In this case, they are integers representing a size measured in basic units '`u`'. The arguments called *h1*, *h2*, ..., *hn* stand for horizontal distances where positive means right, negative left. The arguments called *v1*, *v2*, ..., *vn* stand for vertical distances where positive means down, negative up. All these distances are offsets relative to the current location.

Each graphics command directly corresponds to a `troff \D` escape sequence. See Section 5.27 [Drawing Geometric Objects], page 200.

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 GNU 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 GNU extension.

`De` *h v*<line break>
: Draw an outlined ellipse with a horizontal diameter of *h* and a vertical diameter of *v* (both integers in basic units '`u`') with the leftmost point at current position; then move to the rightmost point of the ellipse.

`DF` *color-scheme* [*component* ...]<line break>
: Set fill color for solid drawing objects using different color schemes; the analogous command for setting the color of text, line graphics, and the outline of graphic objects is '`m`'. The color components are specified as integer arguments between 0 and 65535. The number of color components and their meaning vary for the different color schemes. These commands are generated by GNU `troff`'s escape sequences '`\D'F ...'`' and `\M` (with no other corresponding graphics commands). No position changing. This command is a GNU extension.

  `DFc` *cyan magenta yellow*<line break>
  : Set fill color for solid drawing objects using the CMY color scheme, having the 3 color components *cyan*, *magenta*, and *yellow*.

  `DFd`<line break>
  : Set fill color for solid drawing objects to the default fill color value (black in most cases). No component arguments.

  `DFg` *gray*<line break>
  : Set fill color for solid drawing objects to the shade of gray given by the argument, an integer between 0 (black) and 65535 (white).

  `DFk` *cyan magenta yellow black*<line break>
  : Set fill color for solid drawing objects using the CMYK color scheme, having the 4 color components *cyan*, *magenta*, *yellow*, and *black*.

  `DFr` *red green blue*<line break>
  : Set fill color for solid drawing objects using the RGB color scheme, having the 3 color components *red*, *green*, and *blue*.

`Df` *n*<line break>
: The argument *n* must be an integer in the range −32767 to 32767.

  $0 \leq n \leq 1000$
  : 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 command is superseded by '`DFg`'.

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

```
mg 0 0 65535
Df -1
```

sets all colors to blue.

No position changing. This command is a GNU 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 GNU 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 GNU extension.

`Dt` *n*<line break>
: Set the current line thickness to n (an integer in basic units '`u`') if $n > 0$; if $n = 0$ select the smallest available line thickness; if $n < 0$ set the line thickness proportional to the type size (this is the default before the first '`Dt`' command was specified). For historical reasons, the horizontal position is changed by adding the argument to the actual horizontal position, while the vertical position is not changed. Although this doesn't make sense it is kept for compatibility. This command is a GNU extension.

6.2.2.4 Device Control Commands

Each device control command starts with the letter '`x`', followed by a space character (optional or arbitrary space or tab in GNU `troff`) 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, GNU `troff` outputs the initialization command '`x i`' as '`x init`' and the resolution command '`x r`' as '`x res`'.

In the following, the syntax element ‹line break› means a syntactical line break (see Section 6.2.1.1 [Syntax], page 264).

`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 `groff` uses its internal piping mechanism. The input file is not changed by this command. This command is a GNU extension.

`xf` *`pos file`*‹line break›

The '`f`' stands for *font*.

Mount font described in file *file* at position *pos*, a non-negative integer. Recall Section 5.19.3 [Font Mounting Positions], page 145.

`xH` *`n`*‹line break›

The '`H`' stands for *Height*.

Set glyph height to *n* (a positive integer in scaled points '`z`'). AT&T `troff` uses the unit points ('`p`') instead. See Section 6.2.4 [Output Language Compatibility], page 276.

`xi`‹line break›

The '`i`' stands for *init*.

Initialize device. This is the third command of the header.

`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 minimum horizontal motion, and *v* the minimum vertical motion possible with this device; all arguments are positive integers in basic units '`u`' per inch. This is the second command of the header.

`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 device-independent `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 header.

`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 GNU 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 escape sequence `\X`. Line continuation is a GNU extension.

6.2.2.5 Legacy Compressed Encoding

AT&T `troff` primarily emitted glyphs by writing two digits (a motion) followed by a single character corresponding to a glyph. This syntax is less a command itself than a compressed encoding of the `c` and `h` commands.

ddg
: Move right *dd* (exactly two decimal digits) basic units '`u`', then print glyph *g* (represented as a single character).

  In GNU `troff`, arbitrary syntactical space around and within this command is allowed. Only when a preceding command on

the same line ends with an argument of variable length is a separating space obligatory. In AT&T `troff`, large clusters of these and other commands are used, mostly without spaces; this made such output almost unreadable.

For modern high-resolution devices, this command is impractical because the widths of the glyphs have a greater magnitude in basic units than two decimal digits can represent. In GNU `troff`, this optimization is used only for the devices `X75`, `X75-12`, `X100`, and `X100-12`. For other devices, the commands '`t`' and '`u`' produce more readable output.

6.2.3 GNU `troff` Output Examples

This section presents the output GNU `troff` generates from the same input formatted for three different devices. The input is the phrase '`hell world`' piped to GNU `troff` on the command line.

High-resolution device `ps`

We depict the standard output stream of GNU `troff` in its default build configuration and in the absence of an explicit `-T` option.

```
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 placed onto the standard input stream of `grops` to produce 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

# header
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 placed onto the standard input stream of `grotty` to produce its representation as text file.

AT&T `troff` output

Since a video display has lower resolution than modern printers, GNU `troff`'s output for X11 devices can use the legacy compressed encoding.

```
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 in legacy compressed encoding
ch07e07l03lw06w11o07r05l03dh7
n16 0
x trailer
V1100
x stop
```

Place the foregoing into the standard input stream of `xditview` or `gxditview` to display it in an X11 window.

The legacy compressed encoding makes the content of formatted text in AT&T `troff` output almost incomprehenible.

6.2.4 Output Language Compatibility

The page description 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 incorporate a description of it.

`groff`'s page description language is compatible with this specification except in the following aspects.

- AT&T device-independent `troff`'s quasi-device independence is not yet implemented.
- The printing hardware of the early 1980s differed from today's. `groff`'s output device names also differ from those of AT&T `troff`. For example, the PostScript device in AT&T `troff`, `post` (implemented by the driver command `dpost`), has a resolution of only 720 units per inch, suitable for printers of decades past. `groff`'s `ps` device has a resolution of 72000 units per inch. In principle, by implementing a rescaling mechanism, `groff` could come to emulate AT&T's `post` device.
- While the B-spline command '`D~`' is reliably interpreted by `groff`'s page description language parser, some output drivers don't implement drawing routines for it.
- In GNU `troff`, the argument to the commands '`s`' and '`x H`' uses an implicit unit of scaled points '`z`' whereas AT&T `troff` uses spacing points '`p`'. This isn't an incompatibility, but a compatible extension, for both units coincide for any device without a `sizescale` directive in its `DESC` file, including all postprocessors from AT&T and `groff`'s text (`nroff`-mode) devices. `groff` devices that use `sizescale` 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 drawing position after the commands '`Dp`', '`DP`', and '`Dt`' are processed is illogical. Since old versions of GNU `troff` had this wart, we've retained it for compatibility, but may change it in the future. Wrap these drawing commands with the `\Z` escape sequence to both overcome the illogical positioning and keep your input working consistently regardless of the wart's presence in the implementation.

Appendix A Copying This Manual

A.1 GNU General Public License

Version 3, 29 June 2007

Copyright © 2007 Free Software Foundation, Inc. http://fsf.org/

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Preamble

The GNU General Public License is a free, copyleft license for software and other kinds of works.

The licenses for most software and other practical works are designed to take away your freedom to share and change the works. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change all versions of a program—to make sure it remains free software for all its users. We, the Free Software Foundation, use the GNU General Public License for most of our software; it applies also to any other work released this way by its authors. You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for them if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs, and that you know you can do these things.

To protect your rights, we need to prevent others from denying you these rights or asking you to surrender the rights. Therefore, you have certain responsibilities if you distribute copies of the software, or if you modify it: responsibilities to respect the freedom of others.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must pass on to the recipients the same freedoms that you received. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

Developers that use the GNU GPL protect your rights with two steps: (1) assert copyright on the software, and (2) offer you this License giving you legal permission to copy, distribute and/or modify it.

For the developers' and authors' protection, the GPL clearly explains that there is no warranty for this free software. For both users' and authors' sake, the GPL requires that modified versions be marked as changed, so that their problems will not be attributed erroneously to authors of previous versions.

Some devices are designed to deny users access to install or run modified versions of the software inside them, although the manufacturer can do so. This is fundamentally incompatible with the aim of protecting users' freedom to change the software. The systematic pattern of such abuse occurs in the area of products for individuals to use, which is precisely where it is most unacceptable. Therefore, we have designed this version of the GPL to prohibit the practice for those products. If such problems arise substantially in other domains, we stand ready to extend this provision to those domains in future versions of the GPL, as needed to protect the freedom of users.

Finally, every program is threatened constantly by software patents. States should not allow patents to restrict development and use of software on general-purpose computers, but in those that do, we wish to avoid the special danger that patents applied to a free program could make it effectively proprietary. To prevent this, the GPL assures that patents cannot be used to render the program non-free.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS

0. Definitions.

   "This License" refers to version 3 of the GNU General Public License.

   "Copyright" also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

   "The Program" refers to any copyrightable work licensed under this License. Each licensee is addressed as "you". "Licensees" and "recipients" may be individuals or organizations.

   To "modify" a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a "modified version" of the earlier work or a work "based on" the earlier work.

   A "covered work" means either the unmodified Program or a work based on the Program.

   To "propagate" a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.

   To "convey" a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

   An interactive user interface displays "Appropriate Legal Notices" to the extent that it includes a convenient and prominently visible feature that

(1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion.

1. Source Code.

   The "source code" for a work means the preferred form of the work for making modifications to it. "Object code" means any non-source form of a work.

   A "Standard Interface" means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.

   The "System Libraries" of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A "Major Component", in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.

   The "Corresponding Source" for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work's System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.

   The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source.

   The Corresponding Source for a work in source code form is that same work.

2. Basic Permissions.

   All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to

run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you.

Conveying under any other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary.

3. Protecting Users' Legal Rights From Anti-Circumvention Law.

   No covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures.

   When you convey a covered work, you waive any legal power to forbid circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work's users, your or third parties' legal rights to forbid circumvention of technological measures.

4. Conveying Verbatim Copies.

   You may convey verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.

   You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee.

5. Conveying Modified Source Versions.

   You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:

a. The work must carry prominent notices stating that you modified it, and giving a relevant date.
b. The work must carry prominent notices stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement in section 4 to "keep intact all notices".
c. You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it.
d. If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so.

A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an "aggregate" if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation's users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate.

6. Conveying Non-Source Forms.

You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:

a. Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange.
b. Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge.

c. Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b.

d. Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements.

e. Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at no charge under subsection 6d.

A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A "User Product" is either (1) a "consumer product", which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, "normally used" refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

"Installation Information" for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source. The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part

of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM).

The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying.

7. Additional Terms.

"Additional permissions" are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission.

Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms:

a. Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or

b. Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or

c. Prohibiting misrepresentation of the origin of that material, or re-

quiring that modified versions of such material be marked in reasonable ways as different from the original version; or

d. Limiting the use for publicity purposes of names of licensors or authors of the material; or

e. Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or

f. Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors.

All other non-permissive additional terms are considered "further restrictions" within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.

If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.

Additional terms, permissive or non-permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way.

8. Termination.

You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).

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, you do not qualify to receive new licenses for the same material under section 10.

9. Acceptance Not Required for Having Copies.

   You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance. However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so.

10. Automatic Licensing of Downstream Recipients.

    Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.

    An "entity transaction" is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party's predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.

    You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it.

11. Patents.

    A "contributor" is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor's "contributor version".

    A contributor's "essential patent claims" are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further

modification of the contributor version. For purposes of this definition, "control" includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.

Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor's essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.

In the following three paragraphs, a "patent license" is any express agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To "grant" such a patent license to a party means to make such an agreement or commitment not to enforce a patent against the party.

If you convey a covered work, knowingly relying on a patent license, and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a publicly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so available, or (2) arrange to deprive yourself of the benefit of the patent license for this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients. "Knowingly relying" means you have actual knowledge that, but for the patent license, your conveying the covered work in a country, or your recipient's use of the covered work in a country, would infringe one or more identifiable patents in that country that you have reason to believe are valid.

If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it.

A patent license is "discriminatory" if it does not include within the scope of its coverage, prohibits the exercise of, or is conditioned on the non-exercise of one or more of the rights that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that contain the covered work, unless

you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.

Nothing in this License shall be construed as excluding or limiting any implied license or other defenses to infringement that may otherwise be available to you under applicable patent law.

12. No Surrender of Others' Freedom.

    If conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from conveying the Program.

13. Use with the GNU Affero General Public License.

    Notwithstanding any other provision of this License, you have permission to link or combine any covered work with a work licensed under version 3 of the GNU Affero General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue to apply to the part which is the covered work, but the special requirements of the GNU Affero General Public License, section 13, concerning interaction through a network will apply to the combination as such.

14. Revised Versions of this License.

    The Free Software Foundation may publish revised and/or new versions of the GNU General Public 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.

    Each version is given a distinguishing version number. If the Program specifies that a certain numbered version of the GNU General Public License "or any later version" applies to it, you have the option of following the terms and conditions either of that numbered version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of the GNU General Public License, you may choose any version ever published by the Free Software Foundation.

    If the Program specifies that a proxy can decide which future versions of the GNU General Public License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Program.

Later license versions may give you additional or different permissions. However, no additional obligations are imposed on any author or copyright holder as a result of your choosing to follow a later version.

15. Disclaimer of Warranty.

    THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. Limitation of Liability.

    IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

17. Interpretation of Sections 15 and 16.

    If the disclaimer of warranty and limitation of liability provided above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively state the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

```
one line to give the program's name and a brief idea of what it does.
Copyright (C) year name of author

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation, either version 3 of the License, or (at
your option) any later version.

This program is distributed in the hope that it will be useful, but
WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program.  If not, see http://www.gnu.org/licenses/.
```

Also add information on how to contact you by electronic and paper mail.

If the program does terminal interaction, make it output a short notice like this when it starts in an interactive mode:

```
program Copyright (C) year name of author
This program comes with ABSOLUTELY NO WARRANTY; for details type 'show w'.
This is free software, and you are welcome to redistribute it
under certain conditions; type 'show c' for details.
```

The hypothetical commands '`show w`' and '`show c`' should show the appropriate parts of the General Public License. Of course, your program's commands might be different; for a GUI interface, you would use an "about box".

You should also get your employer (if you work as a programmer) or school, if any, to sign a "copyright disclaimer" for the program, if necessary. For more information on this, and how to apply and follow the GNU GPL, see `http://www.gnu.org/licenses/`.

The GNU General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Lesser General Public License instead of this License. But first, please read `http://www.gnu.org/philosophy/why-not-lgpl.html`.

A.2 GNU Free Documentation License

Version 1.3, 3 November 2008

Copyright © 2000-2018 Free Software Foundation, Inc.
http://fsf.org/

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document *free* in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of

mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

The "publisher" means any person or entity that distributes copies of the Document to the public.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

   You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

   You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

   If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

   If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

   If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque

copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

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

D. Preserve all the copyright notices of the Document.

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

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

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

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If

there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

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

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

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

   You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

   The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

   In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements."

6. COLLECTIONS OF DOCUMENTS

   You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

   You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

   A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

   If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire

aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

   Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

   If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

   You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

   However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

   Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

   Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

10. FUTURE REVISIONS OF THIS LICENSE

    The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in

detail to address new problems or concerns. See `http://www.gnu.org/copyleft/`.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

11. RELICENSING

"Massive Multiauthor Collaboration Site" (or "MMC Site") means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A "Massive Multiauthor Collaboration" (or "MMC") contained in the site means any set of copyrightable works thus published on the MMC site.

"CC-BY-SA" means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

"Incorporate" means to publish or republish a Document, in whole or in part, as part of another Document.

An MMC is "eligible for relicensing" if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

```
Copyright (C)  year  your name.
Permission is granted to copy, distribute and/or modify this document
under the terms of the GNU Free Documentation License, Version 1.3
or any later version published by the Free Software Foundation;
with no Invariant Sections, no Front-Cover Texts, and no Back-Cover
Texts.  A copy of the license is included in the section entitled ``GNU
Free Documentation License''.
```

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with. . . Texts." line with this:

```
with the Invariant Sections being list their titles, with
the Front-Cover Texts being list, and with the Back-Cover Texts
being list.
```

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

Appendix B Indices

B.1 Request Index

Request names appear without a leading control character; the defaults are `.` for the regular control character and `'` for the no-break control character. See Section 5.6.2 [Invoking Requests], page 90.

A

`ab` ... 241
`ad` ... 108
`af` ... 103
`aln` ... 101
`als` ... 174
`am` ... 185
`am1` ... 185
`ami` ... 185
`ami1` ... 185
`as` ... 172
`as1` ... 172
`asciify` ... 221

B

`backtrace` ... 243
`bd` ... 158
`blm` ... 214
`box` ... 218
`boxa` ... 218
`bp` ... 137
`br` ... 107
`break` ... 182
`brp` ... 109

C

`c2` ... 89
`cc` ... 89
`ce` ... 111
`cf` ... 231
`cflags` ... 151
`ch` ... 209
`char` ... 153
`chop` ... 173
`class` ... 155
`close` ... 234
`color` ... 168
`composite` ... 150
`continue` ... 182
`cp` ... 248
`cs` ... 158
`cu` ... 157

D

`da` ... 217
`de` ... 183
`de1` ... 184
`defcolor` ... 168
`dei` ... 185
`dei1` ... 185
`device` ... 235
`devicem` ... 236
`di` ... 217
`do` ... 248
`ds` ... 32, 170
`ds1` ... 170
`dt` ... 212

E

`ec` ... 95
`ecr` ... 95
`ecs` ... 95
`el` ... 178
`em` ... 214
`eo` ... 95
`ev` ... 225
`evc` ... 226
`ex` ... 241

F

fam ... 144
fc ... 128
fchar ... 153
fcolor ... 169
fi ... 107
fl ... 243
fp ... 145
fschar ... 153
fspecial ... 156
ft ... 141
ftr ... 142
fzoom ... 143

G

gcolor ... 169

H

hc ... 115
hcode ... 119
hla ... 120
hlm ... 121
hpf ... 118
hpfa ... 118
hpfcode ... 120
hw ... 114
hy ... 116
hydefault ... 118
hym ... 121
hys ... 121

I

ie ... 178
if ... 178
ig ... 98
in ... 133
it ... 212

K

kern ... 159

L

lc ... 128
length ... 172
lf ... 240
lg ... 159
linetabs ... 127
ll ... 134
ls ... 123
lsm ... 214
lt ... 137

M

mc ... 199
mk ... 191
mso ... 230
msoquiet ... 230

N

na ... 109
ne ... 138
nf ... 108
nh ... 118
nm ... 197
nn ... 198
nop ... 178
nr ... 32, 99, 100, 102
nroff ... 131
ns ... 124
nx ... 231

O

open ... 234
opena ... 234
os ... 139
output ... 221

P

pc ... 137
pchar ... 241
pcolor ... 241
pcomposite ... 241
pev ... 242
pfp ... 242
pftr ... 242
phw ... 242
pi ... 233
pl ... 136
pline ... 242
pm ... 242
pn ... 136
pnr ... 242
po ... 132
ps ... 163
psbb ... 237
pso ... 230
pstream ... 243
pvs ... 166
pwh ... 243

R

rchar ... 154
rd ... 232
return ... 186
rfschar ... 154
rhw ... 114
rj ... 112
rm ... 174
rn ... 174
rnn ... 101
rr ... 101
rs ... 124
rt ... 191

S

schar ... 153
shc ... 115
shift ... 187
sizes ... 164
so ... 230
soquiet ... 230
sp ... 122
special ... 156
spreadwarn ... 244
ss ... 113
stringdown ... 174
stringup ... 174
sty ... 144
substring ... 173
sv ... 139
sy ... 233

T

ta ... 125
tag ... 236
taga ... 236
tc ... 127
ti ... 133
tkf ... 159
tl ... 136
tm ... 240
tm1 ... 240
tmc ... 240
tr ... 129
trf ... 231
trin ... 129
trnt ... 130
troff ... 131

U

uf ... 157
ul ... 157
unformat ... 222

V

vpt ... 206
vs ... 164

W

warn ... 244
warnscale ... 244
wh ... 206
while ... 181
write ... 234
writec ... 234
writem ... 234

B.2 Escape Sequence Index

The escape character, \ by default, is always followed by at least one more input character, making an escape *sequence*. Any token *X* with *X* not in the list below emits a warning and interpolates character *X*. Note the entries for \., which may be obscured by the leader dots, and for \RET and \SPC, which are sorted alphabetically, not by code point order. See Section 5.6.4 [Using Escape Sequences], page 93.

\ 93, 149
\! 220
\" 97
\# 98
\$ 186
\$* 187
\$^ 187
\$@ 187
\$0 188
\% 115
\& 160
\' 151
\(..... 149
\) 162
* 170
\, ... 160
\- 151
\. 189
\/ 160
\: 115
\? 220
\[..... 149
\^ 195
_ 151
\` 151
\\ 189
\{ 179
\} 179
\| 194
\~ 107
\0 195
\a 128
\A 87
\b 204
\B 86
\c 135
\C 150
\d 194
\D 202
\e 94
\E 190
\f 141
\F 144
\g 105
\h 194
\H 156
\k 196
\l 201
\L 201
\m 169
\M 169
\n 102
newline 134
\N 151
\o 196
\O 227
\p 109
\r 194
\R 99, 100
\RET 134
\s 163
space 194
\S 157
\SPC 194
\t 126
\u 194
\v 193
\V 235
\w 195
\x 123
\X 235
\Y 236
\z 196
\Z 196

B.3 Operator Index

See Section 5.4 [Numeric Expressions], page 82.

!

`!` 84

%

`%` 82

&

`&` 83

(

`(` 84

)

`)` 84

*****

`*` 82

+

`+` 82
`+` (unary) 84

–

`-` 82
`-` (unary) 84

/

`/` 82

:

`:` 83

;

`;` 83

<

`<` 83
`<=` 83
`<?` 83

=

`=` 83
`==` 83

>

`>` 83
`>=` 83
`>?` 83

|

`|` 84

B.4 Register Index

Where not used by the formatter itself, a register's associated macro package or program appears in brackets after the register's name.

Interpolate a register name of exactly one character `x` with '`\nx`'; of exactly two characters `xx` with '`\n(xx`'; or of any length `xxx` with '`\n[xxx]`'. See Section 5.8 [Registers], page 99.

$

`$$` 228

%

`%` 137

.

`.$` 186
`.a` 123
`.A` 106
`.b` 158
`.br` 90
`.c` 106
`.cdp` 227
`.ce` 111
`.cht` 227
`.color` 168
`.cp` 248
`.csk` 227
`.C` 248
`.d` 218
`.ev` 225
`.f` 145
`.fam` 144
`.fn` 141
`.fp` 145
`.fzoom` 143
`.F` 106
`.g` 106
`.h` 219
`.height` 156
`.hla` 120
`.hlc` 121
`.hlm` 121
`.hy` 116
`.hydefault` 118
`.hym` 121
`.hys` 121
`.H` 81
`.i` 133
`.in` 133
`.int` 135
`.itm` 212
`.j` 108
`.k` 196
`.kern` 159
`.l` 134
`.lg` 159
`.linetabs` 127
`.ll` 134
`.lt` 137
`.L` 123
`.m` 169
`.M` 169
`.n` 227
`.ne` 210
`.nm` 197
`.nn` 198
`.ns` 124
`.o` 132
`.O` 228
`.p` 136
`.pe` 210
`.pn` 136
`.ps` 166
`.psr` 167
`.pvs` 166
`.P` 106
`.rj` 112
`.R` 106
`.s` 163
`.slant` 157
`.sr` 167
`.ss` 113
`.sss` 113
`.sty` 144
`.t` 208
`.tabs` 125
`.trap` 210
`.trunc` 210
`.T` 106
`.u` 107
`.ul` 157

.U ... 106
.v ... 164
.vpt ... 206
.V ... 81
.w ... 227
.warn ... 244
.x ... 106
.y ... 106
.Y ... 106
.z ... 218

C

c ... 106
ct ... 195

D

DD [ms] ... 37
DI [ms] ... 37
dl ... 219
dn ... 219
dw ... 229
dy ... 229

F

FF [ms] ... 36
FI [ms] ... 36
FM [ms] ... 33
FPD [ms] ... 37
FPS [ms] ... 36
FVS [ms] ... 36

G

GROWPS [ms] ... 35
GS [ms] ... 63

H

HM [ms] ... 33
HORPHANS [ms] ... 35
hours ... 229
hp ... 196
HY [ms] ... 34

L

LL [ms] ... 32
llx ... 237
lly ... 237
ln ... 197
lsn ... 214
lss ... 214
LT [ms] ... 32

M

MINGW [ms] ... 37
minutes ... 229
mo ... 229

N

nl ... 139

O

opmaxx ... 227
opmaxy ... 227
opminx ... 227
opminy ... 227

P

PD [ms] ... 34
PI [ms] ... 34
PO [ms] ... 32
PORPHANS [ms] ... 35
PS [ms] ... 34
PSINCR [ms] ... 35

Q

QI [ms] ... 34

R

rsb ... 195
rst ... 195

S

sb 195
seconds 228
skw 195
slimit 244
ssc 195
st 195
systat 233

T

TC-MARGIN [ms] 37

U

urx 237
ury 237

V

VS [ms] 34

Y

year 229
yr 229

B.5 Macro Index

The package or program with which a macro is associated appears in brackets after the macro's name. They appear without a leading control character (normally '.'). See Section 5.6.3 [Calling Macros], page 92.

[

[[ms] 53

]

] [ms] 53

1

1C [ms] 57

2

2C [ms] 57

A

AB [ms] 39
AE [ms] 39
AI [ms] 38
AM [ms] 64
AU [ms] 38

B

B [ms] 45
B1 [ms] 51
B2 [ms] 51
BD [ms] 51
BI [ms] 45
BT [man] 25
BX [ms] 45

C

CD [ms] 52
CT [man] 26
CW [man] 26
CW [ms] 45

D

DA [ms] 38
De [man] 26
DE [ms] 52
Ds [man] 26
DS [ms] 51, 52

E

EE [man] 26
EF [ms] 56
EH [ms] 56
EN [ms] 53
EQ [ms] 53
EX [man] 26

F

FE [ms] 54
FS [ms] 54

G

G [man] 26
GL [man] 26

H

HB [man] 26

I

I [ms] 45
ID [ms] 51
IP [ms] 41

K

KE [ms] 50
KF [ms] 50
KS [ms] 50

L

LD [ms] 51
LG [ms] 46
LP [ms] 40

M

MC [ms] 57
MS [man] 26

N

ND [ms] 39
NE [man] 27
NH [ms] 42
NL [ms] 46
NT [man] 26

O

OF [ms] 56
OH [ms] 56

P

P1 [ms] 56
PE [ms] 53
PF [ms] 53
Pn [man] 27
PN [man] 27
PP [ms] 40
PS [ms] 53
PT [man] 25
PX [ms] 58

Q

QE [ms] 41
QP [ms] 41
QS [ms] 41

R

R [man] 27
R [ms] 45
RD [ms] 52
RE [ms] 49
RN [man] 27
RP [ms] 38
RS [ms] 49

S

SH [ms] 44
SM [ms] 46

T

TA [ms] 57
TB [man] 26
TC [ms] 58
TE [ms] 52
TL [ms] 38
TS [ms] 52

U

UL [ms] 45

V

VE [man] 27
VS [man] 27

X

XA [ms] 58
XE [ms] 58
XH [ms] 59
XH-REPLACEMENT [ms] 59
XH-UPDATE-TOC [ms] 59
XM [ms] 61
XN [ms] 59
XN-INIT [ms] 59
XN-REPLACEMENT [ms] 59
XP [ms] 41
XR [ms] 60
XS [ms] 58

B.6 String Index

The macro package or program with which a string is associated appears in brackets after the string's name. The formatter itself defines only one string, `.T`.

Interpolate a string name of exactly one character `x` with '`\*x`'; of exactly two characters `xx` with '`\*(xx`'; or of any length `xxx` with '`\*[xxx]`'. See Section 5.22 [Strings], page 170.

!

! [ms] 65

'

' [ms] 63, 64

*****

* [ms] 54

,

, [ms] 64

—

- [ms] 40

.

. [ms] 65
.T 170

/

/ [ms] 64

:

: [ms] 64

<

< [ms] 46

>

> [ms] 46

?

? [ms] 65

^

^ [ms] 64

_

_ [ms] 64

`

` [ms] 63, 64

{

{ [ms] 46

}

} [ms] 46

~

~ [ms] 64

3

3 [ms] 65

8

8 [ms] 65

A

ABSTRACT [ms] 55
Ae [ms] 65
ae [ms] 65

C

C [ms] 64
CF [ms] 33
CH [ms] 33

D

d- [ms] 65
D- [ms] 65

F

FAM [ms] 34
FR [ms] 37

L

LF [ms] 33
LH [ms] 33

M

MONTH1 [ms] 56
MONTH10 [ms] 56
MONTH11 [ms] 56
MONTH12 [ms] 56
MONTH2 [ms] 56
MONTH3 [ms] 56
MONTH4 [ms] 56
MONTH5 [ms] 56
MONTH6 [ms] 56
MONTH7 [ms] 56
MONTH8 [ms] 56
MONTH9 [ms] 56

O

o [ms] 65
oe [ms] 65
OE [ms] 65

Q

q [ms] 65
Q [ms] 40

R

REFERENCES [ms] 55
RF [ms] 33
RH [ms] 33

S

SN [ms] 43
SN-DOT [ms] 43
SN-NO-DOT [ms] 43
SN-STYLE [ms] 36, 43

T

Th [ms] 65
th [ms] 65
TOC [ms] 55

U

U [ms] 40

V

v [ms] 64

B.7 File Keyword Index

See Section 6.1 [Device and Font Description Files], page 257.

#

`#` 257, 260

—

`---` 261

B

`biggestfont` 260

C

`charset` 260, 261
`charset-range` 263

F

`family` 141, 257
`fonts` 146, 156, 257

H

`hor` 258

I

`image_generator` 258

K

`kernpairs` 263

L

`ligatures` 261

N

`name` 261

P

`paperlength` 258
`papersize` 258
`paperwidth` 258
`pass_filenames` 258
`postpro` 258
`prepro` 259
`print` 259

R

`res` 259

S

`sizes` 259
`sizescale` 259
`slant` 261
`spacewidth` 261
`spare1` 260
`spare2` 260
`special` 158, 261
`styles` 141, 144, 259

T

`tcommand` 259

U

`unicode` 259
`unitwidth` 259
`unscaled_charwidths` 259
`use_charnames_in_special` 259

V

`vert` 260

B.8 Program and File Index

A

an.tmac 25

C

changebar 200
chem 7
composite.tmac 150
cs.tmac 119

D

de.tmac 119
DESC 141, 144, 146, 151, 156
diffmk 200

E

ec.tmac 76
en.tmac 119
eqn 7, 52
es.tmac 119

F

fr.tmac 119
freeeuro.pfa 76

G

gdiffmk 200
grn 7
groff 7

H

hyphen.cs 117
hyphen.den 117
hyphen.det 117
hyphen.en 117
hyphen.es 117
hyphen.fr 117
hyphen.it 117
hyphen.pl 117
hyphen.ru 117
hyphen.sv 117
hyphenex.cs 117
hyphenex.en 117
hyphenex.pl 117

I

it.tmac 119

J

ja.tmac 119

K

koi8-r.tmac 75

L

latin1.tmac 75
latin2.tmac 76
latin5.tmac 76
latin9.tmac 76

M

makeindex 23
man.local 25
man.tmac 25
man.ultrix 26

N

nrchbar 200

P

pic 7, 52
pl.tmac 119
post-grohtml 12
pre-grohtml 12
preconv 7

R

rap 7
refer 7, 52
ru.tmac 119

S

soelim 7, 240
sv.tmac 119

T

tbl 7, 52
trace.tmac 186
troff 7
troffrc 11, 119, 120, 131
troffrc-end 11, 120, 131
tty.tmac 131, 132

V

vtroff 176

Z

zh.tmac 119

B.9 Concept Index

!

! complementation operator, multiple logical, in conditional expressions, incompatibility with AT&T troff .. 253

"

", as delimiter 96
", at end of sentence 69, 152
", at the start of a request argument .. 229
", embedding in a macro argument 92

%

%, as delimiter 96

&

&, as delimiter 96

'

', as a comment 98
', as delimiter 96
', at end of sentence 69, 152

(

(, as delimiter 96

)

), as delimiter 96
), at end of sentence 69, 152

*

*, as delimiter 96
*, at end of sentence 69, 152

+

+, and page motion 84
+, as delimiter 96

—

-, and page motion 84
-, as delimiter 96

.

., as delimiter 96
.h register, difference from nl 219
.ps register, compared to .psr 167
.s register, compared to .sr 167
.S register, other troffs' name for .tabs 126
.t register, and diversions 212
.tabs register, other troffs' name for (.S) 126
.V register, and vs 165

/

/, as delimiter 96

:

:, as delimiter 96

<

<, as delimiter 96

=

=, as delimiter 96

>

>, as delimiter 96

[

[, macro names starting with, and refer 87

]

], as part of an identifier 87
], at end of sentence 69, 152
], macro names starting with, and refer 87

\

\!, and copy mode 220
\!, and output request 221
\!, and trnt 130
\!, as delimiter 96
\!, in top-level diversion 221
\!, incompatibilities with AT&T troff . 255
\", interpretation in copy mode 97
\#, interpretation in copy mode 98
\$, interpretation in copy mode 186
\%, and translations 129
\%, as delimiter 96
\%, following \X or \Y 115
\%, in device extension commands 235
\&, and glyph definitions 153
\&, and translations 129
\&, as delimiter 96
\&, at end of sentence 68
\&, in device extension commands 235
\', and translations 129
\', as delimiter 96
\(, and translations 129
\), as delimiter 96
\), in device extension commands 235
*, and warnings 246
*, incompatibilities with AT&T troff . 248
*, interpretation in copy mode 170
\, disabling (eo) 95
\, embedding in a macro argument 92
\,, as delimiter 96
\- glyph, and cflags 152
\-, and translations 129
\-, as delimiter 96
\., interpretation in copy mode 189
\/, as delimiter 96
\:, as delimiter 96
\:, in device extension commands 235
\?, and copy mode 177, 220
\?, as delimiter 96
\?, in top-level diversion 221
\?, incompatibilities with AT&T troff . 255
\?, interpretation in copy mode 220
\[, and translations 129
\^, as delimiter 96
_, and translations 129
_, as delimiter 96
\`, and translations 129
\`, as delimiter 96
\\, as quotation character 189
\\, interpretation in copy mode 189
\{, as delimiter 96
\}, as delimiter 96
\|, as delimiter 96
\~, and translations 129
\~, as delimiter 96
\~, difference from \SPC 92
\~, incompatibilities with AT&T troff . 252
\0, as delimiter 96
\a, and copy mode 128
\a, and translations 129
\a, as delimiter 96
\b, limitations of 204
\c, as delimiter 96
\c, when filling disabled 135
\c, when filling enabled 135
\C, and translations 129
\d, as delimiter 96
\D, delimiters allowed by 96
\e, and glyph definitions 153
\e, and translations 129
\e, as delimiter 96
\e, incompatibilities with AT&T troff . 255
\e, interpretation in copy mode 94
\E, as delimiter 96
\f escape sequence, untokenized on input 142
\f, and font translations 142
\f, incompatibilities with AT&T troff . 250
\F escape sequence, untokenized on input 144
\F, and changing fonts 141
\g, interpretation in copy mode 105
\h, delimiters allowed by 96
\H escape sequence, untokenized on input 156
\H, delimiters allowed by 96
\H, incompatibilities with AT&T troff . 250
\H, using + and - with 84
\H, with fractional type sizes 166
\l, and glyph definitions 153
\l, delimiters allowed by 96
\L, and glyph definitions 153
\L, delimiters allowed by 96
\m escape sequence, untokenized on input 169
\M escape sequence, untokenized on input 170
\n, and warnings 246
\n, incompatibilities with AT&T troff . 248
\n, interpretation in copy mode 102
\N, and translations 129
\N, delimiters allowed by 96
\p, as delimiter 96
\r, as delimiter 96

\R escape sequence, untokenized on input . . . 100
\R, and warnings . . . 246
\R, delimiters allowed by . . . 96
\R, difference from nr . . . 102
\R, using + and - with . . . 84
\RET, interpretation in copy mode . . . 135
\s escape sequence, untokenized on input . . . 164
\s, delimiters allowed by . . . 96
\s, incompatibilities with AT&T troff . 250
\s, using + and - with . . . 84
\s, with fractional type sizes . . . 166
\S escape sequence, untokenized on input . . . 157
\S, delimiters allowed by . . . 96
\S, incompatibilities with AT&T troff . 250
\SPC, as delimiter . . . 96
\SPC, difference from \~ . . . 92
\t, and copy mode . . . 126
\t, and translations . . . 129
\t, and warnings . . . 246
\t, as delimiter . . . 96
\u, as delimiter . . . 96
\v, delimiters allowed by . . . 96
\v, internal representation . . . 238
\V, and copy mode . . . 235
\V, interpretation in copy mode . . . 235
\x, delimiters allowed by . . . 96
\X, followed by \% . . . 115
\Y, followed by \% . . . 115

|

|) operator, use with sp request . . . 123
|, and page motion . . . 84
|, as delimiter . . . 96

A

ab request, incompatibilities with AT&T troff . . . 253
abort (ab) . . . 241
absolute *(sic)* position operator (|) . . . 84
abstract font style . . . 141
abstract font style, setting up (sty) . . . 144
accent marks [ms] . . . 63
access to postprocessor . . . 235
accessing unnamed glyphs with \N . . . 261
activating kerning (kern) . . . 159
activating ligatures (lg) . . . 159
activating track kerning (tkf) . . . 159
ad request, and hyphenation margin . . 121
ad request, and hyphenation space . . . 121
addition . . . 82
additional inter-sentence space . . . 113
adjustment (introduction) . . . 19
adjustment and filling, manipulating . . 106
adjustment mode register (.j) . . . 108
adjustment to both margins, difference from AT&T troff . . . 252
adjustment, and break warnings . . . 245
Adobe Glyph List (AGL) . . . 148
advance width . . . 158
alias, diversion, creating (als) . . . 174
alias, diversion, removing (rm) . . . 175
alias, macro, creating (als) . . . 174
alias, macro, removing (rm) . . . 175
alias, register, creating (aln) . . . 101
alias, register, removing (rr) . . . 101
alias, string, creating (als) . . . 174
alias, string, removing (rm) . . . 175
aliasing fonts with third argument to fp request . . . 145
als request, and \$0 . . . 188
am, am1, ami requests, and warnings . . . 246
annotation, output line . . . 197
appending to a diversion (da, boxa) . . . 217
appending to a file (opena) . . . 234
appending to a macro (am) . . . 185
appending to a string (as) . . . 172
approximation output register (.A) . . . 106
arc, drawing ('\D'a ...') . . . 202
argument . . . 72
arguments to macros . . . 92
arguments to macros, and tabs . . . 90
arguments to requests . . . 90
arguments to requests, and tabs . . . 90
arguments, and compatibility mode . . . 239
arguments, file name, to requests, in other implementations . . . 252
arguments, to escape sequences, delimiting . . . 96
arguments, to strings . . . 170
arithmetic operators . . . 82
artificial fonts . . . 156
as and as1 requests, arguments starting with double quote ", and comments . 172
as request, and comments . . . 171
as, as1 requests, and warnings . . . 246
as1 request, and comments . . . 171
ASCII output encoding . . . 12
asciify request, and writem . . . 234
assertion (arithmetic operator) . . . 82

assign input line number request (lf) . 240
assign number format to register (af) . 103
assignments, indirect 102
assignments, nested 102
AT&T ms, macro package differences . . . 61
AT&T troff bug, in cf request 231
AT&T troff bugs 231
attributes, character cell 141
auto-incrementation of a register 102
automatic font mounting 142
automatic hyphenation 114
automatic hyphenation parameters . . . 116
auxiliary macro package 25
available glyphs, list of (*groff_char*(7) man page) 148
available registers, number of, register (.R) 106

B

background 1
background color name register (.M) . . 169
backslash glyph, formatting (\[rs]) . . . 95
backslash, embedding in a macro argument 92
backslash, printing (\\, \e, \E, \[rs]) . 255
backspace character 194
backspace character, and translations . 129
backtrace of input stack (backtrace) . 243
baseline rule special character(\[ru]) . 201
baseline, text 79, 163
basic scaling unit (u) 80
basic units 79
basic units, conversion to 80
basics of macro package usage 19
bd request, and font styles 144
bd request, and font translations 142
bd request, incompatibilities with AT&T troff 254
beginning diversion (di, box) 217
beginning of conditional block (\{) . . . 179
blank line 70
blank line macro (blm) 70, 91, 214
blank line trap (blm) 91
blank line traps 214
blank lines, disabling 124
block paragraphs 21
block, conditional, beginning (\{) 179
block, conditional, end (\}) 179
blocks, conditional 179
body, of a macro 73
body, of a while request 181
boldface, imitating (bd) 158
bottom margin 207
boundary-relative measurement operator (|) 84
boundary-relative measurement operator (|), use with sp request 123
bounding box 237
box (diversion operation) 218
box request, and warnings 246
box rule special character (\[br]) 201
box, boxa requests, and warnings 246
boxa request, and dn (dl) 219
boxa request, and warnings 246
boxes [ms] 51
bp request, and top-level diversion 137
bp request, and traps (.pe) 210
bp request, causing implicit break 106
bp request, incompatibilities with AT&T troff 253
bp request, using + and - with 84
br glyph, and cflags 152
br request, nilpotence with no-break control character 106
brace escape sequences (\{, \}) 179
break 70, 106
break (introduction) 19
break request, in a while loop 182
break, page 79, 137, 211
break, page (introduction) 21
break, page, final 215
break, page, prevented by vpt 206
breaking file names (\:) 115
breaking URLs (\:) 115
breaking without hyphens (\:) 115
brp request, nilpotence with no-break control character 106
bug, in AT&T troff cf request 231
built-in registers 105
bulleted list, example markup [ms] 47

C

c scaling unit 80
calling macros 92
calling macros (introduction) 72
capabilities of GNU troff 2
case-transforming a string (stringdown, stringup) 174
categories, warning 245
ce request, causing implicit break 106
ce request, difference from '.ad c' 111
cell, character, attributes 141

centered text (filled) 108
centered text (unfilled) 111
centering lines (ce) 111
centering lines (introduction) 21
centimeter scaling unit (c) 80
cf request, and copy mode 231
cf request, arguments starting with double quote ", and comments 229
cf request, causing implicit break 106
cf request, incompatibilities with AT&T troff 252
changing control characters 89
changing font family (fam, \F) 144
changing fonts (ft, \f) 141
changing format, and read-only registers 104
changing the font height (\H) 156
changing the font slant (\S) 157
changing the page number character (pc) 137
changing trap location (ch) 209
changing type sizes (ps, \s) 163, 167
changing vertical line spacing (vs) 164
char request, and comments 153
char request, and soft hyphen character 115
char request, and translations 129
char request, used with \N 151
character 146
character cell attributes 141
character class (class) 155
character class name space, shared with special characters 88
character classes 154
character mappings, composite, dumping (pcomposite) 241
character properties (cflags) 151
character translations 129
character, backspace 194
character, backspace, and translations . 129
character, control (.) 71
character, control, changing (cc) 89
character, defining (char) 153
character, defining fallback (fchar, fschar, schar) 153
character, distinguished from glyph . . . 146
character, dummy (\&) 160
character, dummy (\&), as control character suppressor 71
character, dummy (\&), effect on \l escape sequence 200
character, dummy (\&), effect on kerning 159
character, escape, changing (ec) 95
character, escape, while defining glyph . 153
character, field delimiting (fc) 128
character, field padding (fc) 128
character, horizontal tab 71
character, hyphenation (\%) 115
character, indexed, formatting (\N) . . . 151
character, leader 71
character, leader repetition (lc) 128
character, leader, and translations 129
character, leader, non-interpreted (\a) . 128
character, margins (mc) 199
character, named (\C) 150
character, newline, and translations . . . 129
character, no-break control (') 71
character, no-break control, changing (c2) 89
character, ordinary 86
character, removing definition (rchar, rfschar) 154
character, soft hyphen, setting (shc) . . 115
character, special 129
character, tab repetition (tc) 127
character, tab, and translations 129
character, tab, non-interpreted (\t) . . . 126
character, transparent 152
character, transparent dummy (\)) . . . 162
characters, end-of-sentence 152
characters, end-of-sentence transparent . 69
characters, hyphenation 152
characters, input, and output glyphs, compatibility with AT&T troff 254
characters, invalid for trf request 231
characters, invalid input 74
characters, overlapping 152
characters, special 69
characters, special, list of (*groff_char*(7) man page) 148
characters, unnamed, accessing with \N 261
circle, filled, drawing (‘\D'C ...'') 202
circle, outlined, drawing (‘\D'c ...'') . 202
circle, solid, drawing (‘\D'C ...'') 202
circle, stroked, drawing (‘\D'c ...'') . 202
class of characters (class) 155
classes, character 154
clearing input line trap (it, itc) 212
closing brace escape sequence (\}) 179
closing file (close) 234
code, hyphenation (hcode) 119

color name, background, register (.M) . 169
color name, fill, register (.M) 169
color name, stroke, register (.m) 169
color, default 169
color, fill 168
color, stroke 168
colors 168
colors, defined, dumping (pcolor) 241
command prefix 13
command-line options 7
comments 97
comments in device description files . . . 257
comments in font description files 260
comments, after character definitions . 153
comments, after file name arguments . 119
comments, after file name or system command arguments 229
comments, lining up with tabs 97
comments, with string definitions and appendments 171
comments, with string length measurements 172
common features 21
common name space of macros, diversions, and strings 88
common name space of special characters and character classes 88
comparison expression, output 176
comparison of strings 177
comparison operators 83
compatibility mode 247, 248
compatibility mode, and parameters . . 239
complementation operators !, multiple logical, in conditional expressions, incompatibility with AT&T troff . . 253
complementation, logical 84
composite characters mappings, dumping (pcomposite) 241
composite glyph names 148
conditional block, beginning (\{) 179
conditional block, end (\}) 179
conditional blocks 179
conditional expressions 175
conditional output for terminal (TTY) . 176
conditional page break (ne) 138
conditionals and loops 175
configuring control characters 89
configuring the page length (pl) 136
consecutive hyphenated lines (hlm) . . . 121
constant glyph spacing (cs) 158
contents, table of 23, 128
continuation, input line (\RET) 134
continuation, output line (\c) 135
continue request, in a while loop 182
continued output line register (.int) . . 135
continuous underlining (cu) 157
control character (.) 71
control character, changing (cc) 89
control character, no-break (') 71
control character, no-break, changing (c2) 89
control character, no-break, invoking sp request with, while not on a page . . 123
control characters 89
control line 71
control, line 134
control, page 137
conventions for input 76
conversion to basic units 80
copy mode 188
copy mode, and \! 220
copy mode, and \? 177, 220
copy mode, and \a 128
copy mode, and \t 126
copy mode, and \V 235
copy mode, and cf request 231
copy mode, and device request 235
copy mode, and length request 172
copy mode, and macro parameters 187
copy mode, and output request 221
copy mode, and trf request 231
copy mode, and write request 234
copy mode, and writec request 234
copy mode, and writem request 234
copying environment (evc) 226
correction between upright and slanted glyph (\/, \,) 160
correction, italic (\/) 160
correction, left italic (\,) 160
corrections between slanted and upright glyphs (\/, \,) 160
cover page in [ms], example markup 39
cp request, and glyph definitions 153
cq glyph, at end of sentence 69, 152
creating alias of register (aln) 101
creating alias, for diversion (als) 174
creating alias, for macro (als) 174
creating alias, for string (als) 174
creating new characters (char) 153
credits 5
cs request, and font styles 144
cs request, and font translations 142
cs request, incompatibilities with AT&T troff 254

cs request, with fractional type sizes . . 166
CSTR #54 errata . 101, 113, 132, 138, 157, 163, 174, 195, 229
CSTR #54 erratum, \s escape sequence 163
CSTR #54 erratum, \S escape sequence 157
CSTR #54 erratum, bp request 138
CSTR #54 erratum, po request 132
CSTR #54 erratum, ps request 163
CSTR #54 erratum, rm request 174
CSTR #54 erratum, rr request 101
CSTR #54 erratum, sb register 195
CSTR #54 erratum, ss request 113
CSTR #54 erratum, st register 195
CSTR #54 erratum, yr register 229
current directory 15
current input file name register (.F) . . 106
current page number (%) 138
current time, hours (hours) 229
current time, minutes (minutes) 229
current time, seconds (seconds) 228
customizing man package 25
customizing mdoc package 27

D

da request, and dn (dl) 219
da request, and warnings 246
date, day of the month register (dy) . . 229
date, day of the week register (dw) 229
date, month of the year register (mo) . . 229
date, year register (year, yr) 229
day of the month register (dy) 229
day of the week register (dw) 229
dd glyph, at end of sentence 69, 152
de request, and while 181
de, de1, dei requests, and warnings . . . 246
debugging 240
debugging page location traps 207
decimal point, as delimiter 96
decrementation, automatic, of a register 102
default color 169
default tab stops 125
default units 81
deferred output 204
defined colors, dumping (pcolor) 241
defining character (char) 153
defining character class (class) 155
defining fallback character (fchar, fschar, schar) 153
defining glyph (char) 153
defining symbol (char) 153
delimiters, for escape sequence arguments 96
delimiting character, for fields (fc) 128
delimiting escape sequence arguments . . 96
depth, interpolation 93, 96, 250
depth, nesting, of escape sequences in macro definitions 189
depth, nesting, of interpolations . 93, 96, 250
depth, nesting, of macro definitions . . . 183
depth, of last glyph (.cdp) 227
DESC file format 257
DESC file, and font mounting 146
description file, device, introduced 15
description file, font 140
description file, font, introduced 15
device description file, introduced 15
device description files, comments 257
device request, and copy mode 235
device request, arguments starting with double quote ", and comments 235
device resolution 79, 259
device resolution, obtaining in the formatter 80
devices for output 3
dg glyph, at end of sentence 69, 152
di request, and warnings 246
differences in implementation 248
digit-width space (\0) 195
digits, as delimiters 96
dimensions, line 131
directories for macro packages 14
directory, current 15
directory, device and font description . . 15
directory, for tmac files 14
directory, home 15
directory, platform-specific 15
directory, site-local 15
disabling \ (eo) 95
disabling hyphenation (\%) 115
discardable horizontal space 113
displays 23
displays [ms] 51
displays, and footnotes [ms] 54
distance to next vertical position trap register (.t) 208
diversion 204
diversion name register (.z) 218
diversion name space, shared with macros and strings 88
diversion trap, setting (dt) 212

diversion traps 211
diversion, appending to (da, boxa) 217
diversion, beginning (di, box) 217
diversion, creating alias of (als) 174
diversion, dumping (pm) 242
diversion, ending (di, box) 217
diversion, nested 218
diversion, removing (rm) 174
diversion, removing alias of (rm) 175
diversion, renaming (rn) 174
diversion, stripping final newline 224
diversion, top-level 217
diversion, top-level, and \! 221
diversion, top-level, and \? 221
diversion, top-level, and bp 137
diversion, unformatting (asciify) 221
diversion, vertical drawing position in or on page, register (.d) 218
diversions 217, 222
diversions, and traps 211
division by zero 82
division, truncating 82
dl register, and da (boxa) 219
dn register, and da (boxa) 219
document description macros, [ms] 38
document formats 24
documents, multi-file 240
documents, structuring the source of ... 91
dot, as delimiter 96
double quote, at the start of a request argument 229
double quote, embedding in a macro argument 92
double-spacing (ls) 123
double-spacing (vs, pvs) 165
down-casing a string (stringdown) 174
drawing a filled circle (‘\D'C ...'') ... 202
drawing a filled ellipse (‘\D'E ...'') .. 202
drawing a filled polygon (‘\D'P ...''). 203
drawing a line (‘\D'l ...'') 202
drawing a solid circle (‘\D'C ...'') ... 202
drawing a solid ellipse (‘\D'E ...'') .. 202
drawing a solid polygon (‘\D'P ...''). 203
drawing a spline (‘\D'~ ...'') 202
drawing a stroked circle (‘\D'c ...''). 202
drawing a stroked ellipse (‘\D'e ...''). 202
drawing a stroked polygon (‘\D'p ...'') 203
drawing an arc (‘\D'a ...'') 202
drawing an outlined circle (‘\D'c ...'') 202
drawing an outlined ellipse (‘\D'e ...'') 202
drawing an outlined polygon (‘\D'p ...'') 203
drawing horizontal lines (\l) 201
drawing position 79
drawing position, initial 79
drawing position, vertical (nl) 139
drawing position, vertical, on page or in diversion, register (.d) 218
drawing requests 200
drawing vertical lines (\L) 201
ds and ds1 request, and leading spaces . 171
ds and ds1 requests, arguments starting with double quote ", and comments . 171
ds request, and comments 171
ds, ds1 requests, and warnings 246
ds1 request, and comments 171
dt request, incompatibilities with AT&T troff 253
dummy character (\&) 160
dummy character (\&), as control character suppressor 71
dummy character (\&), effect on \l escape sequence 200
dummy character (\&), effect on kerning 159
dummy character, transparent (\)) ... 162
dummy environment, used by \w escape sequence 195
dumping composite character mappings (pcomposite) 241
dumping defined colors (pcolor) 241
dumping environments (pev) 242
dumping font translations (pftr) 242
dumping hyphenation exception words (phw) 242
dumping macros, strings, or diversions (pm) 242
dumping occupied font mounting positions (pfp) 242
dumping open streams (pstream) 243
dumping page location traps (pwh) 243
dumping pending output line node list (pline) 242
dumping registers (pnr) 242
dumping symbol table (pm) 242

E

effects, font . . . 156
ejection, page . . . 79, 137, 211
ejection, page, of final page . . . 215
ejection, page, prevented by vpt . . . 206
ellipse, filled, drawing (‘\D'E ...'') . . . 202
ellipse, outlined, drawing (‘\D'e ...'') . 202
ellipse, solid, drawing (‘\D'E ...'') . . . 202
ellipse, stroked, drawing (‘\D'e ...'') . 202
em glyph, and cflags . . . 152
em scaling unit (m) . . . 81
embolding of special fonts . . . 158
empty line . . . 70
en scaling unit (n) . . . 81
enabling vertical position traps (vpt) . 206
encoding, input, ISO Latin-1 (8859-1) . . 75
encoding, input, ISO Latin-2 (8859-2) . . 76
encoding, input, ISO Latin-5 (8859-9) . . 76
encoding, input, ISO Latin-9 (8859-15) . 76
encoding, input, KOI8-R . . . 75
encoding, output, ASCII . . . 12
encoding, output, ISO Latin-1 (8859-1) . 12
encoding, output, ISO 646 . . . 12
encoding, output, UTF-8 . . . 12
end of conditional block (\}) . . . 179
end-of-input macro (em) . . . 214
end-of-input trap, setting (em) . . . 214
end-of-input traps . . . 214
end-of-sentence characters . . . 68, 152
end-of-sentence detection, cancellation, on AT&T troff . . . 252
end-of-sentence transparent characters . 69
ending diversion (di, box) . . . 217
endnotes . . . 23
environment . . . 204
environment availability and naming, incompatibilities with . . . 253
environment number/name register (.ev) . . . 225
environment variables . . . 13
environment, copying (evc) . . . 226
environment, dimensions of last glyph (.w, .cht, .cdp, .csk) . . . 227
environment, dummy, used by \w escape sequence . . . 195
environment, previous line length (.n) . 227
environment, switching (ev) . . . 225
environments . . . 224
environments, dumping (pev) . . . 242
equality operator . . . 83
equation example [ms] . . . 53
equations [ms] . . . 52
escape character, as a syntactical quotation character . . . 189
escape character, changing (ec) . . . 95
escape character, formatting (\e) . . . 94
escape character, while defining glyph . 153
escape sequence . . . 89
escape sequence argument delimiters . . . 96
escape sequences . . . 93
escape sequences, brace (\{, \}) . . . 179
escaping newline characters, in strings . 171
ex request, use in debugging . . . 241
ex request, used with nx and rd . . . 232
example markup, bulleted list [ms] . . . 47
example markup, cover page in [ms] . . . 39
example markup, glossary-style list [ms] . 48
example markup, numbered list [ms] . . . 47
examples of invocation . . . 16
exception file, hyphenation pattern . . . 117
exception words, hyphenation, dumping (phw) . . . 242
exiting (ex) . . . 241
expansion of strings (*) . . . 170
explicit hyphen (\%) . . . 121
explicit hyphenation . . . 114
expression, limitation of logical not in . . 84
expression, order of evaluation . . . 84
expression, output comparison . . . 176
expressions, and register format . . . 105
expressions, and space characters . . . 86
expressions, conditional . . . 175
expressions, numeric . . . 82
extra post-vertical line space (\x) . . . 165
extra post-vertical line space register (.a) . . . 123
extra pre-vertical line space (\x) . . . 165
extra spaces between words . . . 71
extreme values representable with Roman numerals . . . 104
extremum operators (>?, <?) . . . 83

F

f scaling unit . . . 168
factor, zoom, of a font (fzoom) . . . 143
fallback character, defining (fchar, fschar, schar) . . . 153
fallback character, removing definition of (rchar, rfschar) . . . 154
fam request, and changing fonts . . . 141
families, font . . . 143
family, font . . . 140
fchar request, and comments . . . 153

features, common 21
fi request, causing implicit break 106
field delimiting character (fc) 128
field padding character (fc) 128
fields 128
fields, and tabs 125
figure space (\0) 195
figures [ms] 52
file formats 257
file name arguments to requests, in other implementations 252
file names, breaking (\:) 115
file stream, writing to (write, writec) . 234
file, appending to (opena) 234
file, closing (close) 234
file, device description, introduced 15
file, font description 140
file, font description, introduced 15
file, hyphenation pattern or exception . 117
file, inclusion (so) 230
file, macro, search path 14
file, next, read (nx) 231
file, opening (open) 234
files, font 257
fill color 168
fill color name register (.M) 169
fill mode, and \c 135
fill mode, disabling, request (nf) 108
fill mode, enabling, request (fi) 107
filled circle, drawing (‘\D'C ...'') 202
filled ellipse, drawing (‘\D'E ...'') . . . 202
filled polygon, drawing (‘\D'P ...'') . . 203
filling 67
filling (introduction) 19
filling and adjustment, manipulating . . 106
filling of output, disabling request (nf) . 108
filling of output, enabling request (fi) . 107
filling, and break warnings 245
filling, and inter-sentence space 113
final newline, stripping in diversions . . 224
first-line indentation, of paragraph 133
fl request, causing implicit break 106
floating keep 23
flush pending output line (fl) 243
flushing of output, timing of, incompatibilities with AT&T troff . 255
flushing, of an output line 237
font 140
font aliasing with third argument to fp request 145
font description file 140
font description file format 257
font description file, format 260
font description file, introduced 15
font description files, comments 260
font directory 15
font effects 156
font families 143
font family 140
font family, changing (fam, \F) 144
font file, format 260
font files 257
font for underlining (uf) 157
font height, changing (\H) 156
font magnification request(fzoom) 143
font metrics 140
font mounting position register (.f) . . . 146
font mounting positions 145
font mounting positions, occupied, dumping (pfp) 242
font mounting, automatic 142
font path 15
font slant, changing (\S) 157
font style 140
font style, abstract 141
font style, abstract, setting up (sty) . . 144
font styles 143
font translation (ftr) 142
font translations, dumping (pftr) 242
font, mounting (fp) 145
font, optical size, setting (fzoom) 143
font, previous, selecting (\f[], \fP) . . . 142
font, previous, selecting (ft) 141
font, selection 141
font, special 140
font, text 140
font, unstyled 140
font, zoom factor (fzoom) 143
fonts, artificial 156
fonts, changing (ft, \f) 141
fonts, searching for 15
fonts, special 155
footers 136, 207
footers [ms] 56
footnote mark [ms] 54
footnotes 23
footnotes [ms] 54
footnotes, and displays [ms] 54
footnotes, and keeps [ms] 54
form letters 232
format of font description file 257
format of font description files 260
format of font files 260
format of register (\g) 105

format, paper 16
format, register 99
format, troff output 264
formats, file 257
formatter instructions 89
formatting a backslash glyph (\[rs]) .. 95
formatting a title line (tl) 136
formatting the escape character (\e) ... 94
fp request, and font translations 142
fp request, incompatibilities with AT&T troff 254
fractional point sizes 166, 253
fractional type sizes 166, 253
fractional type sizes in ms macros 62
French spacing 68
fschar request, and comments 153
fspecial request, and font styles 144
fspecial request, and font translations 142
fspecial request, and glyph search order 146
fspecial request, and imitating bold . 158
ft request, and font translations 142
full-service macro package 25

G

geometry, page 79
GGL (groff glyph list) 148, 155
glossary-style list, example markup [ms] . 48
glyph 146
glyph names, composite 148
glyph pile (\b) 204
glyph properties (cflags) 151
glyph, defining (char) 153
glyph, distinguished from character ... 146
glyph, last, dimensions (.w, .cht, .cdp, .csk) 227
glyph, leader repetition (lc) 128
glyph, numbered (\N) 129
glyph, numbered, accessing (\N) 151
glyph, removing definition (rchar, rfschar) 154
glyph, soft hyphen (hy) 115
glyph, tab repetition (tc) 127
glyphs, available, list of (*groff_char*(7) man page) 148
glyphs, output, and input characters, compatibility with AT&T troff 254
glyphs, overstriking (\o) 196
glyphs, unnamed 151
glyphs, unnamed, accessing with \N ... 261
GNU troff capabilities 2
GNU troff, identification register (.g) 106
GNU troff, PID register ($$) 228
GNU troff, process ID register ($$) .. 228
GNU-specific register (.g) 106
graphic renditions 141
greater than (or equal to) operator 83
groff glyph list (GGL) 148, 155
groff invocation 7
groff—what is it? 1
GROFF_BIN_PATH, environment variable . 13
GROFF_COMMAND_PREFIX, environment variable 13
GROFF_ENCODING, environment variable . 13
GROFF_FONT_PATH, environment variable 14, 15
GROFF_TMAC_PATH, environment variable 14, 15
GROFF_TMPDIR, environment variable ... 14
GROFF_TYPESETTER, environment variable 14
grohtml, the program 12

H

hair space (\^) 195
hanging paragraph 133
hcode request, and glyph definitions .. 153
headers 136, 207
headers [ms] 56
headings, run-in 22
heavy (font stroke weight) 140
height, font, changing (\H) 156
height, of last glyph (.cht) 227
high-water mark register (.h) 219
home directory 15
horizontal discardable space 113
horizontal input line position register (hp) 196
horizontal input line position, saving (\k) 196
horizontal line, drawing (\l) 201
horizontal motion (\h) 194
horizontal motion quantum 258
horizontal motion quantum register (.H) 81
horizontal output line position register (.k) 196
horizontal resolution 258
horizontal resolution register (.H) 81
horizontal space (\h) 194

horizontal space, unformatting 224
horizontal tab character 71
Host System Service Access 228
hours, current time (hours) 229
hpf request, and comments 119
hpf request, and hyphenation language . 120
hpfa request, and comments 119
hw request, and hy restrictions 114
hw request, and hyphenation language . 120
hy glyph, and cflags 152
hyphen, explicit (\%) 121
hyphenated lines, consecutive (hlm) ... 121
hyphenating characters 152
hyphenation 69
hyphenation character (\%) 115
hyphenation code (hcode) 119
hyphenation consecutive line count register (.hlc) 121
hyphenation consecutive line limit register (.hlm) 121
hyphenation exception words, adding . 114
hyphenation exception words, dumping (phw) 242
hyphenation exception words, removing 114
hyphenation language register (.hla) . 120
hyphenation margin (hym) 121
hyphenation margin register (.hym) ... 121
hyphenation mode default register (.hydefault) 118
hyphenation mode register (.hy) 116
hyphenation parameters, automatic ... 116
hyphenation pattern and exception files 117
hyphenation patterns (hpf) 118
hyphenation space (hys) 121
hyphenation space adjustment threshold 121
hyphenation space adjustment threshold register (.hys) 121
hyphenation, automatic 114
hyphenation, disabling (\%) 115
hyphenation, explicit 114
hyphenation, incompatibilities with AT&T troff 252
hyphenation, manipulating 114
hyphenation, manual 114

I

i scaling unit 80
identifiers 86
identifiers, undefined 88
ie request, and font translations 142
ie request, operators to use with 175
if request, and font translations 142
if request, and the '!' operator 82
if request, operators to use with 175
if-else 178
if-then 178
imitating boldface (bd) 158
implementation differences 248
implicit line break 70
implicit trap 211
in request, causing implicit break 106
in request, using + and - with 84
inch scaling unit (i) 80
including a file (so) 230
incompatibilities with AT&T troff ... 248
increment value without changing the register 103
incrementation, automatic, of a register 102
indentation (in) 132
indentation, first line, of paragraph ... 133
indentation, of roff source code 91
indentation, temporary 133
indented paragraphs 21
index, in macro package 23
indexed character, formatting (\N) 151
indicator, scaling 80
indirect assignments 102
initial drawing position 79
input and output requests 228
input characters and output glyphs, compatibility with AT&T troff 254
input characters, invalid 74
input conventions 76
input encoding, ISO Latin-1 (8859-1) .. 75
input encoding, ISO Latin-2 (8859-2) .. 76
input encoding, ISO Latin-5 (8859-9) .. 76
input encoding, ISO Latin-9 (8859-15) . 76
input encoding, KOI8-R 75
input file name, current, register (.F) . 106
input level 93, 96, 250
input line continuation (\RET) 134
input line number register (.c, c.) 106
input line number, assignment, request (lf) 240
input line position, horizontal, saving (\k) 196

input line trap, clearing (`it`, `itc`) 212
input line trap, setting (`it`, `itc`) 212
input line traps 212
input line traps and interrupted lines (`itc`) 212
input line, horizontal position, register (`hp`) 196
input line, productive 110
input stack, backtrace (`backtrace`) ... 243
input stack, setting limit 244
input stream, standard, interpolate from (`rd`) 232
input token 237
inserting horizontal space (`\h`) 194
installation 3
instructing the formatter 89
inter-sentence space 68
inter-sentence space size register (`.sss`) 113
inter-sentence space, additional 113
inter-sentence space, supplemental, using dummy character to suppress 161
inter-word spacing, minimum 113
interactive use of GNU `troff` 243
intercepting requests 90
intermediate output 264
interpolating registers (`\n`) 102
interpolation 72
interpolation depth 93, 96, 250
interpolation of strings (`\*`) 170
interpretation mode 189
interrupted line 135
interrupted line register (`.int`) 135
interrupted lines and input line traps (`itc`) 212
introduction 1
invalid characters for `trf` request 231
invalid input characters 74
invocation examples 16
invoking `groff` 7
invoking requests 90
invoking `sp` request with no-break control character while not on a page 123
ISO Latin-1 (8859-1) input encoding ... 75
ISO Latin-1 (8859-1) output encoding .. 12
ISO Latin-2 (8859-2) input encoding ... 76
ISO Latin-5 (8859-9) input encoding ... 76
ISO Latin-9 (8859-15) input encoding .. 76
ISO 646 output encoding 12
italic correction (`\/`) 160

J

justifying text 106

K

keep, floating 23
keeps (introduction) 23
keeps [`ms`] 50
keeps, and footnotes [`ms`] 54
kerning and ligatures 158
kerning enabled register (`.kern`) 159
kerning, activating (`kern`) 159
kerning, track 159
KOI8-R, input encoding 75

L

landscape page orientation 16
language [`ms`] 55
language, `troff` page description 264
last glyph, dimensions (`.w`, `.cht`, `.cdp`, `.csk`) 227
last-requested point size registers (`.psr`, `.sr`) 167
last-requested type size registers (`.psr`, `.sr`) 167
Latin-1 (ISO 8859-1) input encoding ... 75
Latin-1 (ISO 8859-1) output encoding .. 12
Latin-2 (ISO 8859-2) input encoding ... 76
Latin-5 (ISO 8859-9) input encoding ... 76
Latin-9 (ISO 8859-15) input encoding .. 76
layout, line 131
layout, page 136
`lc` request, and glyph definitions 153
leader 23
leader character 71, 127
leader character, and translations 129
leader character, non-interpreted (`\a`) . 128
leader repetition character (`lc`) 128
leaders 127
leading 163
leading space macro (`lsm`) 70
leading space traps 214
leading spaces 70
leading spaces in `ds` and `ds1` argument . 171
leading spaces macro (`lsm`) 214
left italic correction (`\,`) 160
left margin (`po`) 132
length of a string (`length`) 172
length of line (`ll`) 132
length of previous line (`.n`) 227

length of the page, configuring (pl) ... 136
length of title line, configuring (lt) ... 137
length request, and comments 172
length request, and copy mode 172
length request, arguments starting with double quote ", and comments 173
less than (or equal to) operator 83
letters, form 232
level, input 93, 96, 250
level, suppression nesting, register (.O) . 228
lf request, arguments starting with double quote ", and comments 240
lf request, incompatibilities with AT&T troff 252, 254
ligature 146
ligatures and kerning 158
ligatures enabled register (.lg) 159
ligatures, activating (lg) 159
limitations of \b escape sequence 204
line annotation, output 197
line break 106
line break (introduction) 19
line break, output 70
line break, output (introduction) 19
line control 134
line dimensions 131
line indentation (in) 132
line layout 131
line length (ll) 132
line length register (.l) 134
line length, previous (.n) 227
line number, input, register (.c, c.) .. 106
line number, output, register (ln) 197
line numbers, printing (nm) 197
line space, extra post-vertical (\x) 165
line space, extra pre-vertical (\x) 165
line spacing register (.L) 123
line spacing, post-vertical (pvs) 165
line thickness (‘\D't ...’’) 204
line, blank 70
line, drawing (‘\D'l ...’’) 202
line, horizontal, drawing (\l) 201
line, input, continuation (\RET) 134
line, input, horizontal position, register (hp) 196
line, input, horizontal position, saving (\k) 196
line, interrupted 135
line, output, continuation (\c) 135
line, output, horizontal position, register (.k) 196
line, productive input 110
line, vertical, drawing (\L) 201
line-tabs mode 127
lines, blank, disabling 124
lines, centering (ce) 111
lines, centering (introduction) 21
lines, consecutive hyphenated (hlm) ... 121
lines, interrupted, and input line traps (itc) 212
lines, right-aligning (introduction) 21
list of special characters (*groff_char*(7) man page) 148
listing page location traps (pwh) 243
lists 21
ll request, using + and - with 84
localization 119
localization [ms] 55
locating macro files 14
locating macro packages 14
location, vertical, page, marking (mk) . 191
location, vertical, page, returning to marked (rt) 191
logical “and” operator 83
logical “or” operator 83
logical complementation operator 84
logical conjunction operator 83
logical disjunction operator 83
logical not, limitation in expression 84
logical operators 83
long names 248
loops and conditionals 175
lowercasing a string (stringdown) 174
ls request, alternative to (pvs) 166
lt request, using + and - with 84

M

m scaling unit 81
M scaling unit 81
machine units 79
macro 72
macro arguments 92
macro arguments, and compatibility mode 239
macro arguments, and tabs 90
macro body 73
macro file search path 14
macro name register (\$0) 188
macro name space, shared with strings and diversions 88
macro names, starting with [or], and refer 87
macro package 74

macro package directories 14
macro package search path 14
macro package usage, basics of 19
macro package, auxiliary 25
macro package, full-service 25
macro package, introduction 2
macro package, major 25
macro package, minor 25
macro package, structuring the source of . 91
macro packages, search procedure for . . 14
macro, appending to (`am`) 185
macro, creating alias of (`als`) 174
macro, dumping (`pm`) 242
macro, end-of-input (`em`) 214
macro, parameters (`\$`) 187
macro, removing (`rm`) 174
macro, removing alias of (`rm`) 175
macro, renaming (`rn`) 174
macros packages, tutorial for users of . . 19
macros, recursive 181
macros, writing 182
magnification, font, request (`fzoom`) . . . 143
major macro package 25
major version number register (`.x`) . . . 106
`man` macro package 25
`man` macros, customizing headers and footers of 25
`man` macros, Ultrix-specific 26
man pages 5
manipulating filling and adjustment . . 106
manipulating hyphenation 114
manipulating spacing 122
manipulating type size and vertical spacing 163
manual hyphenation 114
manual pages (“man pages”) 5
margin character (`mc`) 199
margin, bottom 207
margin, hyphenation (`hym`) 121
margin, left (`po`) 132
margin, right 132
margin, top 207
mark, footnote [`ms`] 54
mark, high-water, register (`.h`) 219
marking vertical page location (`mk`) . . . 191
maximum operator 83
maximum value representable with Roman numerals 104
`mdoc` macro package 27
`me` macro package 28
measurements 80
measurements, specifying safely 82
metrics, font 140
minimum inter-word spacing 113
minimum operator 83
minimum value representable with Roman numerals 104
minor macro package 25
minor version number register (`.y`) . . . 106
minutes, current time (`minutes`) 229
`mm` macro package 28
mode, compatibility 248
mode, compatibility, and parameters . . 239
mode, constant glyph spacing (`cs`) 158
mode, copy 188
mode, copy, and `\!` 220
mode, copy, and `\?` 177, 220
mode, copy, and `\a` 128
mode, copy, and `\t` 126
mode, copy, and `\V` 235
mode, copy, and `cf` request 231
mode, copy, and `device` request 235
mode, copy, and `length` request 172
mode, copy, and macro parameters . . . 187
mode, copy, and `output` request 221
mode, copy, and `trf` request 231
mode, copy, and `write` request 234
mode, copy, and `writec` request 234
mode, copy, and `writem` request 234
mode, fill, and `\c` 135
mode, fill, and `break` warnings 245
mode, fill, and inter-sentence space . . . 113
mode, fill, disabling, request (`nf`) 108
mode, fill, enabling, request (`fi`) 107
mode, interpretation 189
mode, line-tabs 127
mode, no-fill request (`nf`) 108
mode, no-fill, and `\c` 135
mode, no-space, enabling, request (`ns`) . 124
mode, `nroff` 130
mode, safer . 11, 15, 106, 230, 233, 234, 248
mode, `troff` 130
mode, unsafe . . . 12, 15, 106, 230, 233, 234
modifying requests 90
modulus by zero 82
modulus operator 82
`mom` macro package 28
month of the year register (`mo`) 229
motion operators 84
motion quanta 81
motion quantum, horizontal 258
motion quantum, horizontal, register (`.H`) 81
motion quantum, vertical 260

motion quantum, vertical, register (.V) . 81
motion, horizontal (\h) 194
motion, vertical (\v) 193
motions, page 191
mounting a font (fp) 145
mounting position 140
mounting positions, font 145
mounting positions, occupied by fonts, dumping (pfp) 242
mounting, font, automatic 142
ms document structure 31
ms macro package 28
ms macros, accent marks 63
ms macros, body text 39
ms macros, creating table of contents . . . 58
ms macros, displays 50
ms macros, document control settings . . 32
ms macros, document description 38
ms macros, equations 52
ms macros, figures 52
ms macros, footers 56
ms macros, footnotes 54
ms macros, fractional type sizes in 62
ms macros, groff differences from AT&T 61
ms macros, headers 56
ms macros, headings 42
ms macros, keeps 50
ms macros, language 55
ms macros, lists 47
ms macros, localization 55
ms macros, margins 57
ms macros, multiple columns 57
ms macros, naming conventions 65
ms macros, nested lists 49
ms macros, obtaining typographical symbols 40
ms macros, page layout 56
ms macros, paragraph handling 40
ms macros, references 52
ms macros, special characters 63
ms macros, strings 63
ms macros, tables 52
ms macros, text settings 40
ms macros, Using PDF features 60
mso request, arguments starting with double quote ", and comments 229
msoquiet request, arguments starting with double quote ", and comments 229
multi-file documents 240
multi-line strings 171
multi-page table example [ms] 53
multiple columns [ms] 57
multiple logical complementation operators ! in conditional expressions, incompatibility with AT&T troff . . 253
multiplication 82

N

n scaling unit 81
name space, common, of macros, diversions, and strings 88
name space, common, of special characters and character classes 88
name, background color, register (.M) . 169
name, fill color, register (.M) 169
name, stroke color, register (.m) 169
named character (\C) 150
names, long 248
naming conventions, ms macros 65
ne request, and the .trunc register . . . 210
ne request, comparison with sv 139
need vertical space request (ne) 138
negating register values 101
negation 82
nested assignments 102
nested diversions 218
nested lists [ms] 49
nesting depth, of escape sequences in macro definitions 189
nesting depth, of interpolations . 93, 96, 250
nesting depth, of macro definitions 183
nesting level, suppression, register (.O) . 228
new page request (bp) 137
newline character, and translations . . . 129
newline character, in strings, escaping . 171
newline, final, stripping in diversions . . 224
next file, read (nx) 231
next free font mounting position register (.fp) 146
next page number register (.pn) 136
next page number, assignment request (pn) 136
next trap name register (.trap) 210
nf request, causing implicit break 106
nl register, and .d 218
nl register, difference from .h 219
nm request, using + and - with 84
no-break control character (') 71
no-break control character, changing (c2) 89
no-break control character, invoking sp request with, while not on a page . . 123

no-fill mode request (nf) 108
no-fill mode, and \c 135
no-space mode, enabling, request (ns) . 124
node 237
node list, of pending output line, dumping (pline) 242
nodes, in device extension commands . 235
non-printing break point (\:) 115
normal (font stroke weight) 140
nr request, and warnings 246
nr request, using + and - with 84
nroff mode 130
number format, assigning to register (af) 103
number of available registers register (.R) 106
number, input line, assignment request (lf) 240
number, next page assignment request (pn) 136
number, next page, register (.pn) 136
numbered glyph (\N) 129
numbered glyph, accessing (\N) 151
numbered list, example markup [ms] . . . 47
numbers, line, printing (nm) 197
numeral-width space (\0) 195
numerals, as delimiters 96
numerals, Roman 103
numeric expression, valid 86
numeric expressions 82
nx request, arguments starting with double quote ", and comments 229
nx request, incompatibilities with AT&T troff 252

O

object creation 186
oblique (font shape) 140
occupied font mounting positions, dumping (pfp) 242
offset, page 79
offset, page (po) 132
open request, and safer mode 11
open request, arguments starting with double quote ", and comments 229
open streams, dumping (pstream) 243
opena request, and safer mode 11
opena request, arguments starting with double quote ", and comments 229
opening brace escape sequence (\}) . . . 179
opening file (open) 234
operator, complementation (!), multiple logical, in conditional expressions, incompatibility with AT&T troff . . 253
operator, scaling 83
operators, arithmetic 82
operators, as delimiters 96
operators, comparison 83
operators, extremum (>?, <?) 83
operators, logical 83
operators, motion 84
operators, unary arithmetic 82
optical size of a font, setting the (fzoom) 143
options 7
order of evaluation in expressions 84
ordinary character 86
orientation, landscape 16
origin 79
orphan 139
orphan lines, preventing with ne 138
os request, and no-space mode 139
outlined circle, drawing (‘\D'c ...'') . 202
outlined ellipse, drawing (‘\D'e ...'') . 202
outlined polygon, drawing (‘\D'p ...'') 203
output and input requests 228
output comparison expression 176
output device name string (.T) 12, 170
output device name string (.T), in other implementations 253
output device usage register (.T) 12
output device usage register (.T), incompatibility with AT&T troff . . 253
output devices 3
output encoding, ASCII 12
output encoding, ISO Latin-1 (8859-1) . 12
output encoding, ISO 646 12
output encoding, UTF-8 12
output flushes, timing of, incompatibilities with AT&T troff 255
output format, troff 264
output glyphs, and input characters, compatibility with AT&T troff 254
output line annotation 197
output line break 70
output line break (introduction) 19
output line number register (ln) 197
output line properties 107
output line, continuation (\c) 135
output line, flush pending (fl) 243
output line, horizontal position, register (.k) 196

output line, node list of pending, dumping (pline) 242
output node 237
output request, and \! 221
output request, and copy mode 221
output, filling, disabling request (nf) . . 108
output, filling, enabling request (fi) . . 107
output, intermediate 264
output, suppressing (\O) 227
output, transparent (\!, \?) 220
output, transparent (cf, trf) 231
output, transparent, incompatibilities with AT&T troff 255
output, troff 263
overlapping characters 152
overstriking glyphs (\o) 196

P

p scaling unit 80
P scaling unit 80
package, macro 74
package, macro, auxiliary 25
package, macro, full-service 25
package, macro, introduction 2
package, macro, major 25
package, macro, minor 25
package, macro, search path 14
package, package, structuring the source of 91
packages, macro, tutorial for users of . . . 19
padding character, for fields (fc) 128
page 79
page break 79, 137, 211
page break (introduction) 21
page break, conditional (ne) 138
page break, final 215
page break, prevented by vpt 206
page control 137
page description language, troff 264
page ejection 79, 137, 211
page ejection status register (.pe) 210
page ejection, of final page 215
page ejection, prevented by vpt 206
page footers 207
page headers 207
page layout 136
page layout [ms] 56
page length register (.p) 136
page length, configuring (pl) 136
page location traps 206
page location traps, debugging 207
page location, vertical, marking (mk) . . 191
page location, vertical, returning to marked (rt) 191
page motions 191
page number character (%) 136
page number character, changing (pc) . 137
page number register (%) 138
page number, next, assignment request (pn) 136
page number, next, register (.pn) 136
page offset 79
page offset (po) 132
page orientation, landscape 16
page, geometry of 79
page, new request (bp) 137
page, vertical drawing position on or in diversion, register (.d) 218
paper format 16
paper size 16
paragraph, hanging 133
paragraphs 21
parameter count register (.$) 186
parameters 186
parameters, and compatibility mode . . 239
parameters, macro (\$) 187
parentheses 84
partially collected line 107
path, for font files 15
path, for tmac files 14
pattern and exception files, hyphenation 117
patterns for hyphenation (hpf) 118
pending node list of output line, dumping (pline) 242
pending output line 107
pending output line, flush (fl) 243
pi request, and safer mode 11
pi request, arguments starting with double quote ", and comments 229
pi request, disabled by default 248
pi request, incompatibilities with AT&T troff 252
pica scaling unit (P) 80
PID of GNU troff register ($$) 228
pile, glyph (\b) 204
pl request, using + and - with 84
plain text approximation output register (.A) 8, 106
planting a trap 206
platform-specific directory 15
pm request, incompatibilities with AT&T troff 254

pn request, using + and - with 84
PNG image generation from PostScript 258
po request, using + and - with 84
point scaling unit (p) 80
point size registers (.s, .ps) 163
point size registers, last-requested (.psr, .sr) 167
point sizes, changing (ps, \s) 163, 167
point sizes, fractional 166, 253
point, scaled, scaling unit (s) 80, 166
point, typographical, scaling unit (z) 80, 166
polygon, filled, drawing (‘\D'P ...'') . 203
polygon, outlined, drawing (‘\D'p ...'') 203
polygon, solid, drawing (‘\D'P ...'') . 203
polygon, stroked, drawing (‘\D'p ...'') 203
position of lowest text line (.h) 219
position, absolute *(sic)* operator (|) 84
position, drawing 79
position, horizontal input line, saving (\k) 196
position, horizontal, in input line, register (hp) 196
position, horizontal, in output line, register (.k) 196
position, mounting 140
position, vertical drawing, on page or in diversion, register (.d) 218
positions, font mounting 145
positions, font mounting, occupied, dumping (pfp) 242
post-vertical line spacing 165
post-vertical line spacing register (.pvs) 166
post-vertical line spacing, changing (pvs) 166
postprocessor access 235
postprocessors 3
PostScript, bounding box 237
PostScript, PNG image generation 258
prefix, for commands 13
preprocessors 3
previous font, selecting (\f[], \fP) ... 142
previous font, selecting (ft) 141
previous line length (.n) 227
print current page register (.P) 11
print to the standard error stream (tm, tm1, tmc) 240
printing backslash (\\, \e, \E, \[rs]) . 255
printing line numbers (nm) 197
printing, zero-width (\z, \Z) 196
process ID of GNU troff register ($$) . 228
productive input line 110
properties of characters (cflags) 151
properties of glyphs (cflags) 151
properties of output lines 107
ps request, and constant glyph spacing . 158
ps request, incompatibilities with AT&T troff 253
ps request, using + and - with 84
ps request, with fractional type sizes .. 166
pso request, and safer mode 11
pvs request, using + and - with 84

Q

quanta, motion 81
quantum, horizontal motion 258
quantum, vertical motion 260
quotation, in roff syntax 189
quoting the control character with \\ . 189
quoting the escape character with \\ . 189

R

radicalex glyph, and cflags 152
ragged-left text 108
ragged-right text 108
rc request, and glyph definitions 153
read (interpolate) from standard input stream (rd) 232
read next file (nx) 231
read-only register, changing format ... 104
read-only register, removing 105
recursive macros 181
refer, and macro names starting with [or] 87
reference, troff 67
references [ms] 52
register format 99
register format, in expressions 105
register, assigning number format to (af) 103
register, creating alias of (aln) 101
register, format (\g) 105
register, read-only, removing 105
register, removing (rr) 101
register, removing alias of (rr) 101
register, renaming (rnn) 101
registers 99

registers, available number of, register (.R) ... 106
registers, built-in ... 105
registers, dumping (pnr) ... 242
registers, interpolating (\n) ... 102
registers, setting (nr, \R) ... 99
removing a read-only register ... 105
removing a register (rr) ... 101
removing alias of register (rr) ... 101
removing alias, for diversion (rm) ... 175
removing alias, for macro (rm) ... 175
removing alias, for string (rm) ... 175
removing character definition (rchar, rfschar) ... 154
removing diversion (rm) ... 174
removing macro (rm) ... 174
removing request (rm) ... 174
removing string (rm) ... 174
renaming a register (rnn) ... 101
renaming diversion (rn) ... 174
renaming macro (rn) ... 174
renaming request (rn) ... 174
renaming string (rn) ... 174
renditions, graphic ... 141
request ... 71, 89
request arguments ... 90
request arguments, and compatibility mode ... 239
request arguments, and tabs ... 90
request, removing (rm) ... 174
request, renaming (rn) ... 174
request, undefined ... 97
requests for drawing ... 200
requests for input and output ... 228
requests handling file name arguments, in other implementations ... 252
requests, intercepting ... 90
requests, invoking ... 90
requests, modifying ... 90
resolution, device ... 79, 259
resolution, device, obtaining in the formatter ... 80
resolution, horizontal ... 258
resolution, horizontal, register (.H) ... 81
resolution, vertical ... 260
resolution, vertical, register (.V) ... 81
RET (keycap notation) ... 4
returning to marked vertical page location (rt) ... 191
revision number register (.Y) ... 106
right margin ... 132
right-aligning lines (introduction) ... 21
right-aligning text (rj) ... 112
rivers ... 252
rj request, causing implicit break ... 106
rn glyph, and cflags ... 152
roman glyph, correction after slanted glyph (\/) ... 160
roman glyph, correction before slanted glyph (\,) ... 160
Roman numerals ... 103
Roman numerals, extrema (maximum and minimum) ... 104
rq glyph, at end of sentence ... 69, 152
rt request, using + and - with ... 84
ru glyph, and cflags ... 152
run-in headings ... 22
running system commands ... 233

S

s scaling unit ... 80, 166
safer mode . 11, 15, 106, 230, 233, 234, 248
saturating arithmetic ... 82
saving horizontal input line position (\k) ... 196
scaled point scaling unit (s) ... 80, 166
scaling indicator ... 80
scaling operator ... 83
scaling unit c ... 80
scaling unit f ... 168
scaling unit i ... 80
scaling unit m ... 81
scaling unit M ... 81
scaling unit n ... 81
scaling unit p ... 80
scaling unit P ... 80
scaling unit s ... 80, 166
scaling unit u ... 80
scaling unit v ... 81
scaling unit z ... 80, 166
schar request, and comments ... 153
search path, font ... 15
search procedure for macro packages ... 14
seconds, current time (seconds) ... 228
selecting the previous font (ft) ... 141
sentence space size register (.sss) ... 113
sentence, cancelling detection of end of, on AT&T troff ... 252
sentence-ending punctuation ... 68
sentences ... 68
sequence, escape ... 89
setting diversion trap (dt) ... 212
setting end-of-input trap (em) ... 214

setting input line trap (`it`, `itc`) 212
setting registers (`nr`, `\R`) 99
setting the page length (`pl`) 136
setting up an abstract font style (`sty`) . 144
`shc` request, and translations 129
site-local directory 15
size of sentence space register (`.sss`) .. 113
size of word space register (`.ss`) 113
size, optical, of a font, setting (`fzoom`) . 143
size, paper 16
size, size 163
sizes, fractional 253
sizes, fractional type 166
skew, of last glyph (`.csk`) 227
slant, font, changing (`\S`) 157
slanted (font shape) 140
`so` request, arguments starting with double quote `"`, and comments 229
`so` request, incompatibilities with AT&T `troff` 252
soft hyphen character, setting (`shc`) ... 115
soft hyphen glyph (`hy`) 115
solid circle, drawing (‘`\D'C ...'`’) 202
solid ellipse, drawing (‘`\D'E ...'`’) 202
solid polygon, drawing (‘`\D'P ...'`’) .. 203
`soquiet` request, arguments starting with double quote `"`, and comments 229
`SOURCE_DATE_EPOCH`, environment variable 14
`sp` request, and no-space mode 124
`sp` request, causing implicit break 106
`sp` request, incompatibilities with AT&T `troff` 254
space between sentences 68
space between sentences register (`.sss`) 113
space between words register (`.ss`) ... 113
space character, as delimiter 96
space characters, in expressions 86
space, between sentences 113
space, between words 113
space, discardable, horizontal 113
space, hair (`\^`) 195
space, horizontal (`\h`) 194
space, horizontal, unformatting 224
space, supplemental inter-sentence, using dummy character to suppress 161
space, thin (`\|`) 194
space, trailing, on input lines, difference from AT&T `troff` 252
space, unbreakable (`\~`) 107
space, unbreakable and unadjustable (`\SPC`) 194
space, vertical, unit (`v`) 81
space, width of a digit (numeral) (`\0`) . 195
space, word 113
spaces in character definitions 153
spaces in `ds` and `ds1` argument, leading . 171
spaces in file name arguments 119
spaces in file name or system command arguments 229
spaces in string definitions and appendments 171
spaces in string length measurement .. 172
spaces, in a macro argument 92
spaces, leading and trailing 70
spacing (introduction) 20
spacing, constant glyph (`cs`) 158
spacing, manipulating 122
spacing, vertical 79, 163
spacing, vertical (introduction) 20
`SPC` (keycap notation) 4
special character name space, shared with character classes 88
special characters 69, 129
special characters [`ms`] 63
special characters, in device extension commands 235
special characters, list of (*groff_char*(7) man page) 148
special font 140
special fonts 146, 155, 261
special fonts, emboldening 158
`special` request, and font translations . 142
`special` request, and glyph search order 146
spline, drawing (‘`\D'~ ...'`’) 202
springing a trap 206
`sqrtex` glyph, and `cflags` 152
`ss` request, incompatibilities with AT&T `troff` 254
stack 224
stacking glyphs (`\b`) 204
standard error stream, write to (`tm`, `tm1`, `tmc`) 240
standard input stream, interpolate from (`rd`) 232
stops, tab 71
stream, standard error, write to (`tm`, `tm1`, `tmc`) 240
streams, open, dumping (`pstream`) 243
string arguments 170
string comparison 177

string expansion (*) 170
string interpolation (*) 170
string name space, shared with macros and diversions 88
string, appending (as) 172
string, creating alias of (als) 174
string, dumping (pm) 242
string, length of (length) 172
string, removing (rm) 174
string, removing alias of (rm) 175
string, renaming (rn) 174
strings 170
strings [ms] 63
strings, multi-line 171
stripping final newline in diversions ... 224
stroke color 168
stroke color name register (.m) 169
stroked circle, drawing ('\D'c ...') .. 202
stroked ellipse, drawing ('\D'e ...') . 202
stroked polygon, drawing ('\D'p ...') 203
structuring source code of documents or macro packages 91
sty request, and changing fonts 141
sty request, and font translations 142
style, font 140
style, font, abstract 141
style, font, abstract, setting up (sty) . 144
styles, font 143
substring (substring) 173
subtraction 82
supplemental inter-sentence space 113
supplemental inter-sentence space, using dummy character to suppress 161
suppressing output (\O) 227
suppression nesting level register (.O) . 228
sv request, and no-space mode 139
sv request, incompatibilities with AT&T troff 254
switching environments (ev) 225
sy request, and safer mode 11
sy request, arguments starting with double quote ", and comments 229
sy request, disabled by default 248
sy request, incompatibilities with AT&T troff 252
symbol 146
symbol table, dumping (pm) 242
symbol, defining (char) 153
symbols (characters and glyphs), using . 146
system commands, running 233
system() return value register (systat) 233

T

tab character 71
tab character encoding 125
tab character, and translations 129
tab character, as delimiter 96
tab character, non-interpreted (\t) ... 126
tab repetition character (tc) 127
tab stop settings register (.tabs) 126
tab stops 71
tab stops, default 125
tab, line-tabs mode 127
table of contents 23, 128
table of contents, creating [ms] 58
table, multi-page, example [ms] 53
tables [ms] 52
tabs, and fields 125
tabs, and macro arguments 90
tabs, and request arguments 90
tabs, before comments 97
tagged paragraphs 21
tags, paragraph 21
temporary indentation 133
terminal, conditional output for 176
text baseline 79, 163
text font 140
text line 72
text line, position of lowest (.h) 219
text underlining 156
text, GNU troff processing of 67
text, justifying 106
text, right-aligning (rj) 112
thickness of lines ('\D't ...') 204
thin space (\|) 194
three-part title (tl) 136
ti request, causing implicit break 106
ti request, using + and - with 84
time, current, hours (hours) 229
time, current, minutes (minutes) 229
time, current, seconds (seconds) 228
timing of output flushes, incompatibilities with AT&T troff 255
title length, configuring (lt) 137
title line length register (.lt) 137
title line, formatting (tl) 136
titles 136
tkf request, and font styles 144
tkf request, and font translations 142
tkf request, with fractional type sizes . 166

tl request, and mc ... 199
tmac, directory ... 14
tmac, path ... 14
TMPDIR, environment variable ... 14
token ... 237
top margin ... 207
top-level diversion ... 217
top-level diversion, and \! ... 221
top-level diversion, and \? ... 221
top-level diversion, and bp ... 137
tr request, and glyph definitions ... 153
tr request, and soft hyphen character ... 115
tr request, incompatibilities with AT&T troff ... 254
track kerning ... 159
track kerning, activating (tkf) ... 159
trailing space, on input lines, difference from AT&T troff ... 252
trailing spaces on text lines ... 70
translations of characters ... 129
translations, font, dumping (pftr) ... 242
transparent characters ... 152
transparent dummy character (\)) ... 162
transparent output (\!, \?) ... 220
transparent output (cf, trf) ... 231
transparent output, incompatibilities with AT&T troff ... 255
trap ... 204
trap name, next, register (.trap) ... 210
trap, changing location (ch) ... 209
trap, distance to next vertical position, register (.t) ... 208
trap, diversion, setting (dt) ... 212
trap, end-of-input, setting (em) ... 214
trap, implicit ... 211
trap, input line, clearing (it, itc) ... 212
trap, input line, setting (it, itc) ... 212
trap, planting ... 206
trap, springing ... 206
traps ... 206
traps, and diversions ... 211
traps, blank line ... 214
traps, diversion ... 211
traps, end-of-input ... 214
traps, input line ... 212
traps, input line, and interrupted lines (itc) ... 212
traps, leading space ... 214
traps, page location ... 206
traps, page location, dumping (pwh) ... 243
traps, page location, listing (pwh) ... 243
traps, sprung by bp request (.pe) ... 210
traps, vertical position ... 206
trf request, and copy mode ... 231
trf request, and invalid characters ... 231
trf request, arguments starting with double quote ", and comments ... 229
trf request, causing implicit break ... 106
trin request, and asciify ... 221
troff mode ... 130
troff output format ... 264
troff page description language ... 264
troff, GNU, interactive use of ... 243
troff, GNU, reference ... 67
troff, interactive use of ... 243
troff, output ... 263
troff, reference ... 67
truncated vertical space register (.trunc) ... 210
truncating division ... 82
TTY, conditional output for ... 176
tutorial for macro package users ... 19
type size ... 163
type size registers (.s, .ps) ... 163
type size registers, last-requested (.psr, .sr) ... 167
type sizes, changing (ps, \s) ... 163, 167
type sizes, fractional ... 166, 253
typeface ... 140
typographical point scaling unit (z) . 80, 166
TZ, environment variable ... 14

U

u scaling unit ... 80
uf request, and font styles ... 144
ul glyph, and cflags ... 152
ul request, and font translations ... 142
Ultrix-specific man macros ... 26
unadjustable and unbreakable space (\SPC) ... 194
unary arithmetic operators ... 82
unbreakable and unadjustable space (\SPC) ... 194
unbreakable space (\~) ... 107
undefined identifiers ... 88
undefined request ... 97
underline font (uf) ... 157
underlining (ul) ... 157
underlining, continuous (cu) ... 157
underlining, text ... 156
unformatting diversions (asciify) ... 221
unformatting horizontal space ... 224
Unicode ... 74

unit, scaling, c 80
unit, scaling, f 168
unit, scaling, i 80
unit, scaling, m 81
unit, scaling, M 81
unit, scaling, n 81
unit, scaling, p 80
unit, scaling, P 80
unit, scaling, s 80, 166
unit, scaling, u 80
unit, scaling, v 81
unit, scaling, z 80, 166
units of measurement 80
units, basic 79
units, basic, conversion to 80
units, default 81
units, machine 79
unnamed glyphs 151
unnamed glyphs, accessing with \N ... 261
unsafe mode 12, 15, 106, 230, 233, 234
unstyled font 140
untokenized escape sequence, \f 142
untokenized escape sequence, \F 144
untokenized escape sequence, \H 156
untokenized escape sequence, \m 169
untokenized escape sequence, \M 170
untokenized escape sequence, \R 100
untokenized escape sequence, \s 164
untokenized escape sequence, \S 157
uppercasing a string (stringup) 174
upright (font shape) 140
upright glyph, correction after slanted glyph (\/) 160
upright glyph, correction before slanted glyph (\,) 160
URLs, breaking (\:) 115
user's tutorial 19
using escape sequences 93
using symbols (characters and glyphs) . 146
UTF-8 output encoding 12

V

v scaling unit 81
valid numeric expression 86
value, incrementing without changing the register 103
variables in environment 13
vee 79
vee scaling unit (v) 81
version number, major, register (.x) .. 106
version number, minor, register (.y) .. 106
vertical drawing position (nl) 139
vertical drawing position on page or in diversion register (.d) 218
vertical line drawing (\L) 201
vertical line spacing register (.v) 164
vertical line spacing, changing (vs) ... 164
vertical line spacing, effective value ... 165
vertical motion (\v) 193
vertical motion quantum 260
vertical motion quantum register (.V) .. 81
vertical page location, marking (mk) ... 191
vertical page location, returning to marked (rt) 191
vertical position trap enable register (.vpt) 206
vertical position traps 206
vertical position traps, enabling (vpt) . 206
vertical position, drawing (nl) 139
vertical resolution 260
vertical resolution register (.V) 81
vertical space unit (v) 81
vertical spacing 79, 163
vertical spacing (introduction) 20

W

warning categories 245
warning level (warn) 244
warnings 245
wh request, incompatibilities with AT&T troff 253
what is groff? 1
while request 181
while request, and font translations .. 142
while request, and the '!' operator 82
while request, confusing with br 182
while request, operators to use with .. 175
widow 139
width computation escape sequence (\w) 195
width, advance 158
width, of last glyph (.w) 227
word space 113
word space size register (.ss) 113
word, definition of 67
write request, and copy mode 234
writec request, and copy mode 234
writem request, and copy mode 234
writing macros 182

Y

year, current, register (`year`, `yr`) 229

Z

`z` scaling unit 80, 166
zero, division and modulus by 82
zero-width printing (`\z`, `\Z`) 196
zoom factor of a font (`fzoom`) 143