Troff User's Manual

Joseph F. Ossanna
Brian W. Kernighan

bwk@research.bell-labs.com

Introduction

Troff and *nroff* are text processors that format text for typesetter- and typewriter-like terminals, respectively. They accept lines of text interspersed with lines of format control information and format the text into a printable, paginated document having a user-designed style. *Troff* and *nroff* offer unusual freedom in document styling: arbitrary style headers and footers; arbitrary style footnotes; multiple automatic sequence numbering for paragraphs, sections, etc; multiple column output; dynamic font and point-size control; arbitrary horizontal and vertical local motions at any point; and a family of automatic overstriking, bracket construction, and line-drawing functions.

Troff produces its output in a device-independent form, although parameterized for a specific device; *troff* output must be processed by a driver for that device to produce printed output.

Troff and *nroff* are highly compatible with each other and it is almost always possible to prepare input acceptable to both. Conditional input is provided to enable the user to embed input expressly destined for either program. *Nroff* can prepare output directly for a variety of terminal types and is capable of utilizing the full resolution of each terminal. *Nroff* is the same program as *troff*; in fact, on Plan 9 *nroff* is a shell script that calls *troff* with the −N argument.

Background to the Plan 9 Edition

The primary change to *troff* and *nroff* for Plan 9 is support of the Unicode Standard, which was added during 1992 and 1993. There are two results. First, there is much less need for the myriad of two-character names that are so much a part of *troff* lore; in Plan 9, for example, one naturally uses the Unicode character ½ instead of *troff*'s `\(12`. Second, the output device, though called `utf`, is almost always a form of PostScript printer; the panoply of special drivers for different typesetters has largely disappeared. Unfortunately, not all PostScript printers can cope with Unicode characters, so there remains a need for programs that synthesize PostScript characters from bitmaps; this is especially true for Asian languages.

Background to the Second Edition

Troff was originally written by the late Joe Ossanna in about 1973, in assembly language for the PDP-11, to drive the Graphic Systems CAT typesetter. It was rewritten in C around 1975, and underwent slow but steady evolution until Ossanna's death late in 1977.

In 1979, Brian Kernighan modified *troff* so that it would produce output for a variety of typesetters, while retaining its input specifications. Over the decade from 1979 to

1989, the internals have been modestly revised, though much of the code remains as it was when Ossanna wrote it.

Troff reads parameter files each time it is invoked, to set values for machine resolution, legal type sizes and fonts, and character names, character widths and the like. *Troff* output is ASCII characters in a simple language that describes where each character is to be placed and in what size and font. A post-processor must be written for each device to convert this typesetter-independent language into specific instructions for that device.

The output language contains information that was not readily identifiable in the older output. In the newer language, the beginning of each page, line, and word is marked, so post-processors can do device-specific optimizations such as sorting the data vertically or printing it boustrophedonically, independent of *troff*.

Capabilities for graphics have been added: *troff* recognizes commands for drawing diagonal lines, circles, ellipses, circular arcs, and quadratic B-splines. There are also ways to pass arbitrary information to the output, unprocessed by *troff*.

A number of limitations have been eased or eliminated. A document may have an arbitrary number of fonts on any page (if the output device permits it, of course). Fonts may be accessed merely by naming them; ''mounting'' is no longer necessary. There are no limits on the number of characters. Character height and slant may be set independently of width.

The remainder of this document contains a description of usage and command-line options; a summary of requests, escape sequences, and pre-defined number registers; a reference manual; tutorial examples; and a list of commonly-available characters.

Acknowledgements

Joe Ossanna's *troff* remains a remarkable accomplishment. For more than twenty years, it has proven a robust tool, taking unbelievable abuse from a variety of preprocessors and being forced into uses that were never conceived of in the original design, all with considerable grace under fire.

Recent versions of *troff* have profited from significant code improvements by Jaap Akkerhuis, Dennis Ritchie, Ken Thompson, and Molly Wagner. UTF facilities owe much to Jaap Akkerhuis. Andrew Hume, Doug McIlroy, Peter Nelson and Ravi Sethi made valuable suggestions on the manual. I fear that the remaining bugs are my fault.

Usage

Troff or *nroff* is invoked as

```
troff   options  files
nroff   options  files
```

where *options* represents any of a number of option arguments and *files* represents the list of files containing the document to be formatted. An argument consisting of a single minus '−' represents standard input. If no filenames are given input is taken from the standard input. The options, which may appear in any order so long as they appear before the files, are:

`−m`*name*	Read the macro file `/sys/lib/tmac.`*name* before the input *files*.
`−T`*name*	Specifies the type of the output device. Specific devices are site-dependent. For *troff*, the most useful name is `utf`. For *nroff*, useful names include `37` for the (default) Model 37 Teletype, `lp` for ''dumb'' line printer terminals (no half-line motions, no reverse motions), and `think` for the HP ThinkJet printer.
`−i`	Read standard input after the input files are exhausted.
`−o`*list*	Print only pages whose page numbers appear in *list*, which consists of comma-separated numbers and number ranges. A number range has the form $N-M$ and means pages N through M; a initial $-N$ means from the beginning to page N; and a final $N-$ means from N to the end.
`−n`*N*	Number first generated page N.
`−r`*aN*	Set number register *a* (one-character) to N.
`−s`*N*	Stop every N pages. *Nroff* will halt prior to every N pages (default $N=1$) to allow paper loading or changing, and will resume upon receipt of a newline. *Troff* will include a ''pause'' code every N pages; its meaning, if any, depends on the output device.
`−u`*N*	Set amount of emboldening for the bd request to N.
`−F`*path*	Look in directory *path* for font information; the defaults are `/sys/lib/troff/font` and `/sys/lib/troff/term` for *troff* and *nroff* respectively.
	troff Only
`−a`	Send a printable approximation of the results to the standard output.
	nroff Only
`−e`	Produce equally-spaced words in adjusted lines, using full terminal resolution.
`−h`	Use tabs instead of spaces to speed up printing.
`−q`	Invoke the simultaneous input-output mode of the `rd` request.

Each option is a separate argument; for example,

```
troff −Tutf −ms −mpictures −o4,6,8−10 file1 file2
```

requests formatting of pages 4, 6, and 8 through 10 of a document contained in the files named *file1* and *file2*, specifies the output in UTF, and invokes the macro packages `−ms` and `−mpictures`.

Various pre- and post-processors are available for use with *nroff* and *troff*. These include the equation preprocessor *eqn* (for *troff* only), the table-construction preprocessor *tbl*, and *pic* and *grap* for various forms of graphics.

Request Summary

In the following table, the notation $\pm N$ in the *Request Form* column means that the forms N, $+N$, or $-N$ are permitted, to set the parameter to N, increment it by N, or decrement it by N, respectively. Plain N means that the value is used to set the parameter. *Initial Values* separated by **;** are for *troff* and *nroff* respectively. In the *Notes* column,

B	Request normally causes a break. The use of **'** as control character (instead of **.**) suppresses the break function.
D	Mode or relevant parameters associated with current diversion level.
E	Relevant parameters are a part of the current environment.
O	Must stay in effect until logical output.
P	Mode must be still or again in effect at the time of physical output.
T	*troff* only; no effect in *nroff*.
v, **p**, **m**, **u**	Default scale indicator; if not specified, scale indicators are ignored.

Request Form	*Initial Value*	*If No Argument*	*Notes*	*Explanation*
1. **General Information**				
2. **Font and Character Size Control**				
`.ps` ±*N*	10 point	previous	E,T	Point size; also `\s`±*N*.
`.ss` *N*	12/36**m**	ignored	E,T	Space-character size set to *N*/36 em.
`.cs` *F N M*	off	–	P,T	Constant character space (width) mode (font *F*).
`.bd` *F N*	off	–	P,T	Embolden font *F* by *N*−1 units.
`.bd S` *F N*	off	–	P,T	Embolden Special Font when current font is *F*.
`.ft` *F*	Roman	previous	E	Change to font *F*; also `\f`*x*, `\f(`*xx*, `\f`*N*.
`.fp` *N F L*	R,I,B,...,S	ignored	–	Mount font named *F* on physical position *N*≦1; long name is *L* if given.
3. **Page Control**				
`.pl` ±*N*	11i	11i	**v**	Page length.
`.bp` ±*N*	*N*=1	–	B,**v**	Eject current page; next page number *N*.
`.pn` ±*N*	*N*=1	ignored	–	Next page number *N*.
`.po` ±*N*	1i; 0	previous	**v**	Page offset.
`.ne` *N*	–	*N*=1v	D,**v**	Need *N* vertical space.
`.mk` *R*	none	internal	D	Mark current vertical place in register *R*.
`.rt` ±*N*	none	internal	D,**v**	Return (upward only) to marked vertical place.
4. **Text Filling, Adjusting, and Centering**				
`.br`	–	–	B	Break.
`.fi`	fill	–	B,E	Fill output lines.
`.nf`	fill	–	B,E	No filling or adjusting of output lines.
`.ad` *c*	adj, both	adjust	E	Adjust output lines with mode *c*; *c*=`l`,`r`,`c`,`b`,*none*
`.na`	adjust	–	E	No output line adjusting.
`.ce` *N*	off	*N*=1	B,E	Center next *N* input text lines.
5. **Vertical Spacing**				
`.vs` *N*	12p; 1/6i	previous	E,**p**	Vertical baseline spacing (*V*).
`.ls` *N*	*N*=1	previous	E	Output *N*−1 **v**'s after each text output line.
`.sp` *N*	–	*N*=1v	B,**v**	Space vertical distance *N* in either direction.
`.sv` *N*	–	*N*=1v	**v**	Save vertical distance *N*.
`.os`	–	–	–	Output saved vertical distance.
`.ns`	space	–	D	Turn no-space mode on.
`.rs`	–	–	D	Restore spacing; turn no-space mode off.

6. Line Length and Indenting

.ll ±*N*	6.5i	previous	E,**m**	Line length.
.in ±*N*	*N*=0	previous	B,E,**m**	Indent.
.ti ±*N*	–	ignored	B,E,**m**	Temporary indent.

7. Macros, Strings, Diversion, and Position Traps

.de *xx yy*	–	.*yy*=..	–	Define or redefine macro *xx*; end at call of *yy*.
.am *xx yy*	–	.*yy*=..	–	Append to a macro.
.ds *xx string*	–	ignored	–	Define a string *xx* containing *string*.
.as *xx string*	–	ignored	–	Append *string* to string *xx*.
.rm *xx*	–	ignored	–	Remove request, macro, or string.
.rn *xx yy*	–	ignored	–	Rename request, macro, or string *xx* to *yy*.
.di *xx*	–	end	D	Divert output to macro *xx*.
.da *xx*	–	end	D	Divert and append to *xx*.
.wh *N xx*	–	–	**v**	Set location trap; negative is w.r.t. page bottom.
.ch *xx N*	–	–	**v**	Change trap location.
.dt *N xx*	–	off	D,**v**	Set a diversion trap.
.it *N xx*	–	off	E	Set an input-line count trap.
.em *xx*	none	none	–	End macro is *xx*.

8. Number Registers

.nr *R* ±*N M*		–	**u**	Define and set number register *R*; auto-increment by *M*.
.af *R c*	arabic	–	–	Assign format to register *R* (*c*=1, i, I, a, A).
.rr *R*	–	–	–	Remove register *R*.

9. Tabs, Leaders, and Fields

.ta *Nt* ...	0.5i; 0.8n	none	E,**m**	Tab settings; left-adjusting, unless *t*=R (right), C (centered).
.tc *c*	none	none	E	Tab repetition character.
.lc *c*	.	none	E	Leader repetition character.
.fc *a b*	off	off	–	Set field delimiter *a* and pad character *b*.

10. Input and Output Conventions and Character Translations

.ec *c*	\	\	–	Set escape character.
.eo	on	–	–	Turn off escape character mechanism.
.lg *N*	on; –	on	T	Ligature mode on if *N*>0.
.ul *N*	off	*N*=1	E	Underline (italicize in *troff*) *N* input lines.
.cu *N*	off	*N*=1	E	Continuous underline in *nroff*; in *troff*, like ul.
.uf *F*	Italic	Italic	–	Underline font set to *F* (to be switched to by ul).
.cc *c*	.	.	E	Set control character to *c*.
.c2 *c*	'	'	E	Set no-break control character to *c*.
.tr *abcd....*	none	–	O	Translate *a* to *b*, etc., on output.

11. Local Horizontal and Vertical Motions, and the Width Function

12. Overstrike, Bracket, Line-drawing, Graphics, and Zero-width Functions

13. Hyphenation.

.nh	hyphenate	–	E	No hyphenation.
.hy *N*	hyphenate	hyphenate	E	Hyphenate; *N* = mode.
.hc *c*	\%	\%	E	Hyphenation indicator character *c*.
.hw *word* ...		ignored	–	Add words to hyphenation dictionary.

14. Three-Part Titles.

.tl *'l'c'r'*		–	–	Three-part title; delimiter may be any character.
.pc *c*	%	off	–	Page number character.
.lt ±*N*	6.5i	previous	E,**m**	Length of title.

15. Output Line Numbering.

.nm ±*N M S I*		off	E	Number mode on or off, set parameters.
.nn *N*	–	*N*=1	E	Do not number next *N* lines.

16. Conditional Acceptance of Input

`.if` *c any*	–	–		If condition *c* true, accept *any* as input; for multi-line, use \{*any*\}.
`.if !`*c any*	–	–		If condition *c* false, accept *any*.
`.if` *N any*	–	**u**		If expression $N > 0$, accept *any*.
`.if !`*N any*	–	**u**		If expression $N \leqq 0$ [sic], accept *any*.
`.if` *'s1's2' any*	–	–		If string *s*1 identical to *s*2, accept *any*.
`.if !`*'s1's2' any*	–	–		If string *s*1 not identical to *s*2, accept *any*.
`.ie` *c any*	–	**u**		If portion of if-else; all above forms (like `if`).
`.el` *any*	–	–		Else portion of if-else.

17. Environment Switching

`.ev` *N*	$N=0$	previous	–	Environment switch (push down).

18. Insertions from the Standard Input

`.rd` *prompt*	–	*prompt*=BEL	–	Read insertion.
`.ex`	–	–	–	Exit.

19. Input/Output File Switching

`.so` *filename*		–	–	Switch source file (push down).
`.nx` *filename*		end-of-file	–	Next file.
`.sy` *string*		–	–	Execute program *string*. Output not interpolated.
`.pi` *string*		–	–	Pipe output to program *string*.
`.cf` *filename*		–	–	Copy file contents to *troff* output.

20. Miscellaneous

`.mc` *c N*	–	off	E,**m**	Set margin character *c* and separation *N*.
`.tm` *string*	–	newline	–	Print *string* on terminal (standard error).
`.ab` *string*	–	newline	–	Print *string* on standard error, exit program.
`.ig` *yy*	–	*.yy*= `..`	–	Ignore input until call of *yy*.
`.lf` *N f*		–	–	Set input line number to *N* and filename to *f*.
`.pm` *t*	–	all	–	Print macro names, sizes; if *t* present, print total.
`.fl`	–	–	B	Flush output buffer.

21. Output and Error Messages

22. Output Language

23. Device and Font Description Files

Alphabetical Request and Section Number Cross Reference

ab 20	ce 4	ec 10	ft 2	lg 10	nh 13	pm 20	so 19	tr 10
ad 4	cf 19	el 16	hc 13	lf 20	nm 15	pn 3	sp 5	uf 10
af 8	ch 7	em 7	hw 13	ll 6	nn 15	po 3	ss 2	ul 10
am 7	cs 2	eo 10	hy 13	ls 5	nr 8	ps 2	sv 5	vs 5
as 7	cu 10	ev 17	ie 16	lt 14	ns 5	rd 18	sy 19	wh 7
bd 2	da 7	ex 18	if 16	mc 20	nx 19	rm 7	ta 9	
bp 3	de 7	fc 9	ig 20	mk 3	os 5	rn 7	tc 9	
br 4	di 7	fi 4	in 6	na 4	pc 14	rr 8	ti 6	
c2 10	ds 7	fl 20	it 7	ne 3	pi 19	rs 5	tl 14	
cc 10	dt 7	fp 2	lc 9	nf 4	pl 3	rt 3	tm 20	

Escape Sequences for Characters, Indicators, and Functions

Section Reference	*Escape Sequence*	*Meaning*
10.1	`\\`	\ prevents or delays the interpretation of \
10.1	`\e`	Printable version of the current escape character.
2.1	`\'`	´ (acute accent); equivalent to `\(aa`
2.1	`` \` ``	` (grave accent); equivalent to `\(ga`
2.1	`\-`	– Minus sign in the current font
7.	`\.`	Period (dot) (see `de`)
11.1	*space*	Unpaddable space-size space character
11.1	`\0`	Digit width space
11.1	`|`	1/6 em narrow space character (zero width in *nroff*)
11.1	`\^`	1/12 em half-narrow space character (zero width in *nroff*)
4.1	`\&`	Non-printing, zero width character
10.6	`\!`	Transparent line indicator
10.8	`\"`	Beginning of comment; continues to end of line
13.	`\%`	Default optional hyphenation character
2.1	`\(`*xx*	Character named *xx*
7.1	`\*`*x*, `\*(`*xx*	Interpolate string *x* or *xx*
7.3	`\$`*N*	Interpolate argument $1 \leqq N \leqq 9$
9.1	`\a`	Non-interpreted leader character
12.3	`\b'`*abc...*`'`	Bracket building function
4.2	`\c`	Connect to next input text
2.1	`\C'`*xyz*`'`	Character named *xyz*
11.1	`\d`	Downward 1/2 em vertical motion (1/2 line in *nroff*)
12.5	`\D'`*c...*`'`	Draw graphics function *c* with parameters ...; *c*=`l,c,e,a,~`
2.2	`\f`*x*, `\f(`*xx*, `\f`*N*	Change to font named *x* or *xx*, or position *N*
8.	`\g`*x*, `\g(`*xx*	Format of number register *x* or *xx*
11.1	`\h'`*N*`'`	Local horizontal motion; move right *N* (negative left)
2.3	`\H'`*N*`'`	Height of current font is *N*
11.3	`\k`*x*	Mark horizontal input place in register *x*
12.4	`\l'`*Nc*`'`	Horizontal line drawing function (optionally with *c*)
12.4	`\L'`*Nc*`'`	Vertical line drawing function (optionally with *c*)
8.	`\n`*x*, `\n(`*xx*	Contents of number register *x* or *xx*
2.1	`\N'`*N*`'`	Character number *N* on current font
12.1	`\o'`*abc...*`'`	Overstrike characters *a*, *b*, *c*, ...
4.1	`\p`	Break and spread output line
11.1	`\r`	Reverse 1 em vertical motion (reverse line in *nroff*)
2.3	`\s`*N*, `\s±`*N*	Point-size change function; also `\s(`*nn*, `\s±(`*nn*
2.2	`\S'`*N*`'`	Slant output *N* degrees
9.1	`\t`	Non-interpreted horizontal tab
11.1	`\u`	Reverse (up) 1/2 em vertical motion (1/2 line in *nroff*)
11.1	`\v'`*N*`'`	Local vertical motion; move down N (negative up)
11.2	`\w'`*string*`'`	Width of *string*
5.2	`\x'`*N*`'`	Extra line-space function (negative before, positive after)
10.7	`\X'`*string*`'`	Output *string* as device control function
12.2	`\z`*c*	Print *c* with zero width (without spacing)
16.	`\{`	Begin conditional input
16.	`\}`	End conditional input
10.8	*newline*	Concealed (ignored) newline
–	*Z*	*Z*, any character not listed above

The escape sequences `\\`, `\.`, `\"`, `\$`, `\*`, `\a`, `\n`, `\t`, `\g`, and *newline* are interpreted in copy mode (§7.2).

Predefined Number Registers

Section Reference	*Register Name*	*Description*
3.	%	Current page number.
11.2	ct	Character type (set by \w function).
7.4	dl	Width (maximum) of last completed diversion.
7.4	dn	Height (vertical size) of last completed diversion.
-	dw	Current day of the week (1-7).
-	dy	Current day of the month (1-31).
15.	ln	Output line number.
-	mo	Current month (1-12).
4.1	nl	Vertical position of last printed text baseline.
11.2	sb	Depth of string below baseline (generated by \w function).
11.2	st	Height of string above baseline (generated by \w function).
-	yr	Last two digits of current year.

Predefined Read-Only Number Registers

Section Reference	*Register Name*	*Description*
19.	$$	Process id of *troff* or *nroff*.
7.3	.$	Number of arguments available at the current macro level.
5.2	.a	Post-line extra line-space most recently used in \x'*N*'.
-	.A	Set to 1 in *troff*, if −a option used; always 1 in *nroff*.
2.3	.b	Emboldening level.
20.	.c	Number of lines read from current input file.
7.4	.d	Current vertical place in current diversion; equal to nl, if no diversion.
2.2	.f	Current font number.
20.	.F	Current input file name [sic].
4.	.h	Text baseline high-water mark on current page or diversion.
11.1	.H	Available horizontal resolution in basic units.
6.	.i	Current indent.
4.2	.j	Current ad mode.
4.1	.k	Current output horizontal position.
6.	.l	Current line length.
5.1	.L	Current ls value.
4.	.n	Length of text portion on previous output line.
3.	.o	Current page offset.
3.	.p	Current page length.
7.5	.R	Number of unused number registers.
-	.T	Set to 1 in *nroff*, if −T option used; always 0 in *troff*.
2.3	.s	Current point size.
7.5	.t	Distance to the next trap.
4.1	.u	Equal to 1 in fill mode and 0 in nofill mode.
5.1	.v	Current vertical line spacing.
11.1	.V	Available vertical resolution in basic units.
11.2	.w	Width of previous character.
-	.x	Reserved version-dependent register.
-	.y	Reserved version-dependent register.
7.4	.z	Name [sic] of current diversion.

Reference Manual

1. General Explanation

1.1. Form of input. Input consists of *text lines*, which are destined to be printed, interspersed with *control lines*, which set parameters or otherwise control subsequent processing. Control lines begin with a *control character*—normally **.** (period) or **'** (single quote)—followed by a one or two character name that specifies a basic *request* or the substitution of a user-defined *macro* in place of the control line. The control character **'** suppresses the *break* function—the forced output of a partially filled line—caused by certain requests. The control character may be separated from the request/macro name by white space (spaces and/or tabs) for aesthetic reasons. Names should be followed by either space or newline. Control lines with unrecognized names are ignored.

Various special functions may be introduced anywhere in the input by means of an *escape* character, normally `\`. For example, the function `\n`*R* causes the interpolation of the contents of the *number register R* in place of the function; here *R* is either a single character name as in `\n`*x*, or a two-character name introduced by a left-parenthesis, as in `\n(`*xx*.

1.2. Formatter and device resolution. *Troff* internally stores and processes dimensions in units that correspond to the particular device for which output is being prepared; values from 300 to 1200/inch are typical. See §23. *Nroff* internally uses 240 units/inch, corresponding to the least common multiple of the horizontal and vertical resolutions of various typewriter-like output devices. *Troff* rounds horizontal/vertical numerical parameter input to the actual horizontal/vertical resolution of the output device indicated by the `−T` option (default `post`). *Nroff* similarly rounds numerical input to the actual resolution of its output device (default Model 37 Teletype).

1.3. Numerical parameter input. Both *nroff* and *troff* accept numerical input with the appended scale indicators shown in the following table, where *S* is the current type size in points and *V* is the current vertical line spacing in basic units.

Scale Indicator	Meaning
`i`	Inch
`c`	Centimeter
`P`	Pica = 1/6 inch
`m`	Em = *S* points
`n`	En = Em/2
`p`	Point = 1/72 inch
`u`	Basic unit
`v`	Vertical line space *V*
none	Default, see below

In *nroff*, both the em and the en are taken to be equal to the nominal character width, which is output-device dependent; common values are 1/10 and 1/12 inch. Actual character widths in *nroff* need not be all the same and constructed characters such as -> (→) are often extra wide. The default scaling is `m` for the horizontally-oriented requests and functions `ll`, `in`, `ti`, `ta`, `lt`, `po`, `mc`, `\h`, `\l`, and horizontal coordinates of `\D`; `v` for the vertically-oriented requests and functions `pl`, `wh`, `ch`, `dt`, `sp`, `sv`, `ne`, `rt`, `\v`, `\x`, `\L`, and vertical coordinates of `\D`; `p` for the `vs` request; and `u` for the requests `nr`, `if`, and `ie`. *All* other requests ignore any scale indicators. When a number register containing an already appropriately scaled number is interpolated to provide numerical input, the unit scale indicator `u` may need to be appended to prevent an additional inappropriate default scaling. The number, *N*, may be specified in decimal-fraction form but the parameter finally stored is rounded to an integer number of basic units. Internal computations are performed in integer arithmetic.

The *absolute position* indicator `|` may be prefixed to a number *N* to generate the distance to the vertical or horizontal place *N*. For vertically-oriented requests and functions, `|`*N* becomes the distance in basic units from the current vertical place on the page or in a *diversion* (§7.4) to the vertical place *N*. For *all* other requests and functions, `|`*N* becomes the distance from the current horizontal place on the *input* line to the horizontal place *N*. For example,

```
.sp |3.2c
```

will space in the required direction to 3.2 centimeters from the top of the page.

1.4. Numerical expressions. Wherever numerical input is expected, an expression involving parentheses, the arithmetic operators +, −, /, ∗, % (mod), and the logical operators <, >, <=, >=, = (or ==), & (and), `:` (or) may be used. Except where controlled by parentheses, evaluation of expressions is left-to-right; there is no operator precedence. In the case of certain requests, an initial + or − is stripped and interpreted as an increment or decrement indicator respectively. In the presence of default scaling, the desired scale indicator must be attached to *every* number in an expression for which the desired and default scaling differ. For example, if the number register `x` contains 2 and the current point size is 10, then

```
.ll (4.25i+\nxP+3)/2u
```

will set the line length to 1/2 the sum of 4.25 inches + 2 picas + 3 ems.

1.5. Notation. Numerical parameters are indicated in this manual in two ways. $\pm N$ means that the argument may take the forms *N*, $+N$, or $-N$ and that the corresponding effect is to set the parameter to *N*, to increment it by *N*, or to decrement it by *N* respectively. Plain *N* means that an initial algebraic sign is *not* an increment indicator, but merely the sign of *N*. Generally, unreasonable numerical input is either ignored or truncated to a reasonable value. For example, most requests expect to set parameters to non-negative values; exceptions are `sp`, `wh`, `ch`, `nr`, and `if`. The requests `ps`, `ft`, `po`, `vs`, `ls`, `ll`, `in`, and `lt` restore the previous parameter value in the absence of an argument.

Single character arguments are indicated by single lower case letters and one/two character arguments are indicated by a pair of lower case letters. Character string arguments are indicated by multi-character mnemonics.

2. Font and Character Size Control

2.1. Character set. The *troff* character set is defined by a description file specific to each output device (§23). There are normally several regular fonts and one or more special fonts. Characters are input as themselves, as `\(`*xx*, as `\C'`*name*`'`, or as `\N'`*n*`'`. The form `\C'`*name*`'` permits a name of any length; the form `\N'`*n*`'` refers to the *n*-th character on the current font, whether named or not.

Normally the input characters `‘`, `’`, and `–` are printed as ‘, ’, and ‐ respectively; `\‘`, `\’`, and `\–` produce `, ´, and -. If the character does not exist in the font, *troff* assumes the width is 1 em and outputs the character with a C name as defined in Section 22. (This is independent of how the device handles characters unknown to it.)

Nroff has an analogous, but different, mechanism for defining legal characters and how to print them. By default all characters are valid. There are such additional characters as may be available on the output device, such characters as may be constructed by overstriking or other combination, and those that can reasonably be mapped into other printable characters. The exact behavior is determined by a driving table prepared for each device.

2.2. Fonts. *Troff* begins execution by reading information for a set of defaults fonts, said to be *mounted*; conventionally, the first four are Times Roman (R), Times Italic (I), Times Bold (B), and Times Bold Italic (BI) , and the last is a Special font (S) containing miscellaneous characters. (This document uses Lucida Sans in place of Times.) The set of fonts and positions is determined by the device description file, described in §23.

The current font, initially Roman, may be changed by the `ft` request, or by embedding at any desired point `\f`*x*, `\f(`*xx*, or `\f`*N*, where *x* and *xx* are the name of a font and *N* is a numerical font position.

It is not necessary to change to the Special font; characters on that font are automatically handled as if they were physically part of the current font. The Special font may actually be several fonts; the name S is reserved and is generally used for one of these. All special fonts must be mounted after regular fonts.

Troff can be informed that any particular font is mounted by use of the `fp` request. The list of known fonts is installation dependent. In the subsequent discussion of font-related requests, *F* represents either a one/two-character font name or the numerical font position. The current font is available (as a numerical position) in the read-only number register `.f`.

A request for a named but not-mounted font is honored if the font description information exists. In this way, there is no limit on the number of fonts that may be printed in any part of a document. Mounted fonts may be handled more efficiently, and they may be referred to by their mount positions, but there is no other difference. Mention of an unmounted font loads it temporarily at font position zero, which serves as a one-font cache.

The function `\S'`±*N*`'` causes the current font to be slanted by ±*N* degrees. Not all devices support slanting.

Nroff understands font control and normally underlines italic characters (see §10.5).

2.3. Character size. Character point sizes available depend on the specific output device; a typical (historical) set of values is 6, 7, 8, 9, 10, 11, 12, 14, 16, 18, 20, 22, 24, 28, and 36. This is a range of 1/12 inch to 1/2 inch. The `ps` request is used to change or restore the point size. Alternatively the point size may be changed between any two characters by embedding a `\s`*N* at the desired point to set the size to *N*, or a `\s`±*N* ($1 \leqq N \leqq 9$) to increment/decrement the size by *N*; `\s0` restores the previous size. Requested point size values that are between two valid sizes yield the larger of the two.

Note that through an accident of history, a construction like `\s39` is parsed as size 39, and thus converted to size 36 (given the sizes above), while `\s40` is parsed as size 4 followed by 0. The forms `\s(`*nn* and `\s±(`*nn* permit specification of sizes that would otherwise be ambiguous.

The current size is available in the `.s` register. *Nroff* ignores type size requests.

The function `\H'`±*N*`'` sets the height of the current font to *N*, or increments it by +*N*, or decrements it by −*N*; if *N*=0, the height is restored to the current point size. In each case, the width is unchanged. Not all devices support independent height and width for characters.

Request Form	*Initial Value*	*If No Argument*	*Notes*
`.ps` ±*N**	10 point	previous	E

Point size set to ±*N*. Alternatively, embed `\s`*N* or `\s`±*N*. Any positive size value may be requested; if invalid, the next larger valid size will result, with a maximum of 36. A paired sequence +*N*, −*N* will work because the

*The fields have the same meaning as described earlier in the Request Summary.

previous requested value is also remembered. Ignored in *nroff*.

`.ss` *N* 12/36 em ignored E

Space-character size (i.e., inter-word gap) is set to *N*/36 ems. This size is the minimum word spacing in adjusted text. Ignored in *nroff*.

`.cs` *FNM* off – P

Constant character space (width) mode is set on for font *F* (if mounted); the width of every character will be taken to be *N*/36 ems. If *M* is absent, the em is that of the character's point size; if *M* is given, the em is *M* points. All affected characters are centered in this space, including those with an actual width larger than this space. Special Font characters occurring while the current font is *F* are also so treated. If *N* is absent, the mode is turned off. The mode must be in effect when the characters are physically printed. Ignored in *nroff*.

`.bd` *F N* off – P

The characters in font *F* will be artificially emboldened by printing each one twice, separated by *N*−1 basic units. A reasonable value for *N* is 3 when the character size is near 10 points. If *N* is missing the embolden mode is turned off. The emboldening value *N* is in the `.b` register.

This paragraph is printed with `.bd R 3`. The mode must be in effect when the characters are physically printed. Ignored in *nroff*.

`.bd S` *F N* off – P

The characters in the Special font will be emboldened whenever the current font is *F*. The mode must be in effect when the characters are physically printed. Ignored in *nroff*.

`.ft` *F* Roman previous E

Font changed to *F*. Alternatively, embed `\f`*F*. The font name `P` is reserved to mean the previous font, and the name `S` for the special font.

`.fp` *N F L* R,I,B,...,S ignored –

Font position. This is a statement that a font named *F* is associated with position *N*. It is a fatal error if *F* is not known. For fonts with names longer than two characters, *L* refers to the long name, and *F* becomes a synonym. There is generally a limit of about 10 mounted fonts.

3. Page control

Top and bottom margins are not automatically provided; it is conventional to define two *macros* and to set *traps* for them at vertical positions 0 (top) and −*N* (distance *N* up from the bottom). See §7 and Tutorial Examples §T2. A pseudo-page transition onto the first page occurs either when the first *break* occurs or when the first *non-diverted* text processing occurs. Arrangements for a trap to occur at the top of the first page must be completed before this transition. In the following, references to the *current diversion* (§7.4) mean that the mechanism being described works during both ordinary and diverted output (the former considered as the top diversion level).

The limitations on *troff* and *nroff* output dimensions are device dependent.

`.pl` ±*N* 11 in 11 in **v**

Page length set to ±*N*. The current page length is available in the `.p` register.

`.bp` ±*N* *N*=1 – B,**v**

Begin page. The current page is ejected and a new page is begun. If ±*N* is given, the new page number will be ±*N*. Also see request `ns`.

`.pn` ±*N*	*N*=1	ignored	–

Page number. The next page (when it occurs) will have the page number ±*N*. A `pn` must occur before the initial pseudo-page transition to affect the page number of the first page. The current page number is in the % register.

`.po` ±*N*	1 in; 0	previous	**v**

Page offset. The current *left margin* is set to ±*N*. The *troff* initial value provides 1 inch of paper margin on a typical device. The current page offset is available in the `.o` register.

`.ne` *N*	–	*N*=1 *V*	D,**v**

Need *N* vertical space. If the distance *D* to the next trap position (see §7.5) is less than *N*, a forward vertical space of size *D* occurs, which will spring the trap. If there are no remaining traps on the page, *D* is the distance to the bottom of the page. If $D < V$, another line could still be output and spring the trap. In a diversion, *D* is the distance to the *diversion trap*, if any, or is very large.

`.mk` *R*	none	internal	D

Mark the current vertical place in an internal register (both associated with the current diversion level), or in register *R*, if given. See `rt` request.

`.rt` ±*N*	none	internal	D,**v**

Return *upward only* to a marked vertical place in the current diversion. If ±*N* (with respect to current place) is given, the place is ±*N* from the top of the page or diversion or, if *N* is absent, to a place marked by a previous `mk`. The `sp` request (§5.3) may be used instead of `rt` by spacing to the absolute place stored in a explicit register, e.g., using `.mk` *R* ... `.sp |\n`*R*`u`; this also works when the motion is downwards.

4. Text Filling, Adjusting, and Centering

4.1. Filling and adjusting. Normally, words are collected from input text lines and assembled into a output text line until some word does not fit. An attempt is then made to hyphenate the word to put part of it into the output line. The spaces between the words on the output line are then increased to spread out the line to the current *line length* minus any current *indent*. A *word* is any string of characters delimited by the *space* character or the beginning/end of the input line. Any adjacent pair of words that must be kept together (neither split across output lines nor spread apart in the adjustment process) can be tied together by separating them with the *unpaddable space* character “\ ” (backslash-space). The adjusted word spacings are uniform in *troff* and the minimum interword spacing can be controlled with the `ss` request (§2). In *nroff*, they are normally nonuniform because of quantization to character-size spaces; however, the command line option `−e` causes uniform spacing with full output device resolution. Filling, adjustment, and hyphenation (§13) can all be prevented or controlled. The text length on the last line output is available in the `.n` register, and text baseline position on the page for this line is in the `nl` register. The text baseline high-water mark (lowest place) on the current page is in the `.h` register. The current horizontal output position is in the `.k` register.

An input text line *ending* with `.`, `?`, or `!`, optionally followed by any number of `"`, `'`, `)`, `]`, `*`, or †, is taken to be the end of a sentence, and an additional space character is automatically provided during filling. To prevent this, add `\&` to the end of the input line. Multiple inter-word space characters found in the input are retained, except for trailing spaces; initial spaces also cause a break.

When filling is in effect, a `\p` may be embedded or attached to a word to cause a break at the end of the word and have the resulting output line spread out to fill the current line length.

A text input line that happens to begin with a control character can be made not to look like a control line by prefixing it with the non-printing, zero-width filler character `\&`. Still another way is to specify output translation of some convenient character into the control character using `tr` (§10.5).

4.2. Interrupted text. The copying of a input line in *nofill* (non-fill) mode can be interrupted by terminating the partial line with a `\c`. The next encountered input text line will be considered to be a continuation of the same line of input text. Similarly, a word within *filled* text may be interrupted by terminating the word (and line) with `\c`; the next encountered text will be taken as a continuation of the interrupted word. If the intervening control lines cause a break, any partial line will be forced out along with any partial word.

`.br`	–	–	B

Break. The filling of the line currently being collected is stopped and the line is output without adjustment. Text lines beginning with space characters (but not tabs) and empty text lines (blank lines) also cause a break.

`.fi`	fill on	–	B,E

Fill subsequent output lines. The register `.u` is 1 in fill mode and 0 in nofill mode.

`.nf`	fill on	–	B,E

Nofill. Subsequent output lines are neither filled nor adjusted. Input text lines are copied directly to output lines without regard for the current line length.

`.ad` *c*	adj, both	adjust	E

Line adjustment is begun. If fill mode is not on, adjustment will be deferred until fill mode is back on. If the type indicator *c* is present, the adjustment type is changed as shown in the following table.

Indicator	Adjust Type
`l`	adjust left margin only
`r`	adjust right margin only
`c`	center
`b` or `n`	adjust both margins
absent	unchanged

The number register `.j` contains the current value of the `ad` setting; its value can be recorded and used subsequently to set adjustment.

`.na`	adjust	–	E

Noadjust. Adjustment is turned off; the right margin will be ragged. The adjustment type for `ad` is not changed. Output line filling still occurs if fill mode is on.

`.ce` *N*	off	$N=1$	B,E

Center the next *N* input text lines within the current available horizontal space (line-length minus indent). If $N=0$, any residual count is cleared. A break occurs after each of the *N* input lines. If the input line is too long, it will be left adjusted.

5. Vertical Spacing

5.1. Baseline spacing. The vertical spacing (*V*) between the baselines of successive output lines can be set using the `vs` request. *V* should be large enough to accommodate the character sizes on the affected output lines. For the common type sizes (9–12 points), usual typesetting practice is to set *V* to 2 points greater than the point size; *troff* default is 10-point type on a 12-point spacing (as in this document). The current *V* is available in the `.v` register. Multiple-*V* line separation (e.g., double spacing) may be requested with `ls`, but it is better to use a large `vs` instead; certain preprocessors assume single spacing. The current line spacing is available in the `.L` register.

5.2. Extra line–space. If a word contains a tall construct requiring the output line containing it to have extra vertical space before and/or after it, the *extra–line–space* function `\x'`*N*`'` can be embedded in or attached to that word. If *N* is negative, the output line containing the word will be preceded by *N* extra vertical space; if *N* is positive, the output line containing the word will be followed by *N* extra vertical space. If successive requests for extra space apply to the same line, the maximum values are used. The most recently utilized post-line extra line-space is available in the `.a` register.

In `\x'`*...*`'` and other functions having a pair of delimiters around their parameter, the delimiter choice (here `'`) is arbitrary, except that it can not look like the continuation of a number expression for *N*.

5.3. Blocks of vertical space. A block of vertical space is ordinarily requested using `sp`, which honors the *no–space* mode and which does not space past a trap. A contiguous block of vertical space may be reserved using `sv`.

`.vs` *N*	12pts; 1/6in	previous	E,**p**

Set vertical baseline spacing size *V*. Transient extra vertical space is available with `\x`*'N '* (see above).

`.ls` *N*	*N*=1	previous	E

Line spacing set to ±*N*. *N*−1 *V*s (blank lines) are appended to each output text line. Appended blank lines are omitted, if the text or previous appended blank line reached a trap position.

`.sp` *N*	–	*N*=1 *V*	B,**v**

Space vertically in either direction. If *N* is negative, the motion is backward (upward) and is limited to the distance to the top of the page. Forward (downward) motion is truncated to the distance to the nearest trap. (Recall the use of `.sp |`*N* from §1.3.) If the no-space mode is on, no spacing occurs (see `ns` and `rs` below).

`.sv` *N*	–	*N*=1 *V*	**v**

Save a contiguous vertical block of size *N*. If the distance to the next trap is greater than *N*, *N* vertical space is output. No-space mode has no effect. If this distance is less than *N*, no vertical space is immediately output, but *N* is remembered for later output (see `os`). Subsequent `sv` requests will overwrite any still remembered *N*.

`.os`	–	–	–

Output saved vertical space. No-space mode has no effect. Used to finally output a block of vertical space requested by an earlier `sv` request.

`.ns`	space	–	D

No-space mode turned on. When on, no-space mode inhibits `sp` requests and `bp` requests *without* a next page number. No-space mode is turned off when a line of output occurs, or with `rs`.

`.rs`	space	–	D

Restore spacing. The no-space mode is turned off.

Blank text line.		–	B

Causes a break and output of a blank line exactly like `sp 1`.

6. Line Length and Indenting

The maximum line length for fill mode may be set with `ll`. The indent may be set with `in`; an indent applicable to only the next output line may be set with `ti`. The line length includes indent space but not page offset space. The line length minus the indent is the basis for centering with `ce`. The effect of `ll`, `in`, or `ti` is delayed, if a partially collected line exists, until after that line is output. In fill mode the length of text on an output line is less than or equal to the line length minus the indent. The current line length and indent are available in registers `.l` and `.i` respectively. The length of *three-part titles* produced by `tl` (see §14) is independently set by `lt`.

`.ll` $\pm N$	6.5 in	previous	E,**m**

Line length is set to $\pm N$.

`.in` $\pm N$	N=0	previous	B,E,**m**

Indent is set to $\pm N$. The indent is prefixed to each output line.

`.ti` $\pm N$	–	ignored	B,E,**m**

Temporary indent. The next output text line will be indented a distance $\pm N$ with respect to the current indent. The resulting total indent may not be negative. The current indent is not changed.

7. Macros, Strings, Diversion, and Position Traps

7.1. Macros and strings. A *macro* is a named set of arbitrary *lines* that may be invoked by name or with a *trap*. A *string* is a named string of *characters*, not including a newline character, that may be interpolated by name at any point. Request, macro, and string names share the same name list. Macro and string names may be one or two characters long and may usurp previously defined request, macro, or string names; this implies that built-in operations may be (irrevocably) redefined. Any of these entities may be renamed with `rn` or removed with `rm`.

Macros are created by `de` and `di`, and appended to by `am` and `da`; `di` and `da` cause normal output to be stored in a macro. A macro is invoked in the same way as a request; a control line beginning `.`*xx* will interpolate the contents of macro *xx*. The remainder of the line may contain up to nine *arguments*.

Strings are created by `ds` and appended to by `as`. The strings *x* and *xx* are interpolated at any desired point with `\*`*x* and `\*(`*xx* respectively. String references and macro invocations may be nested.

7.2. Copy mode input interpretation. During the definition and extension of strings and macros (not by diversion) the input is read in *copy mode*. In copy mode, input is copied without interpretation except that:

- The contents of number registers indicated by `\n` are interpolated.
- Strings indicated by `\*` are interpolated.
- Arguments indicated by `\$` are interpolated.
- Concealed newlines indicated by `\`*newline* are eliminated.
- Comments indicated by `\"` are eliminated.
- `\t` and `\a` are interpreted as ASCII horizontal tab and SOH respectively (§9).
- `\\` is interpreted as `\`.
- `\.` is interpreted as “`.`”.

These interpretations can be suppressed by prefixing a `\`. For example, since `\\` maps into a `\`, `\\n` will copy as `\n`, which will be interpreted as a number register indicator when the macro or string is reread.

7.3. Arguments. When a macro is invoked by name, the remainder of the line is taken to contain up to nine arguments. The argument separator is the space character (not tab), and arguments may be surrounded by double quotes to permit embedded space characters. Pairs of double quotes may be embedded in double-quoted arguments to represent a single double-quote character. The argument "" is explicitly null. If the desired arguments won't fit on a line, a concealed newline may be used to continue on the next line. A trailing double quote may be omitted.

When a macro is invoked the *input level* is *pushed down* and any arguments available at the previous level become unavailable until the macro is completely read and the previous level is restored. A macro's own arguments can be interpolated at any point within the macro with \$*N*, which interpolates the *N*th argument ($1 \leqq N \leqq 9$). If an invoked argument does not exist, a null string results. For example, the macro *xx* may be defined by

```
.de xx   \" begin definition
Today is \\$1 the \\$2.
..       \" end definition
```

and called by

```
.xx Monday 14th
```

to produce the text

```
Today is Monday the 14th.
```

Note that each \$ was concealed in the definition with a prefixed \. The number of arguments is in the `.$` register.

No arguments are available at the top (non-macro) level, within a string, or within a trap-invoked macro.

Arguments are copied in copy mode onto a stack where they are available for reference. It is advisable to conceal string references (with an extra \) to delay interpolation until argument reference time.

7.4. Diversions. Processed output may be diverted into a macro for purposes such as footnote processing (see Tutorial §T5) or determining the horizontal and vertical size of some text for conditional changing of pages or columns. A single diversion trap may be set at a specified vertical position. The number registers `dn` and `dl` respectively contain the vertical and horizontal size of the most recently ended diversion. Processed text that is diverted into a macro retains the vertical size of each of its lines when reread in *nofill* mode regardless of the current *V*. Constant-spaced (`cs`) or emboldened (`bd`) text that is diverted can be reread correctly only if these modes are again or still in effect at reread time. One way to do this is to embed in the diversion the appropriate `cs` or `bd` requests with the *transparent* mechanism described in §10.6.

Diversions may be nested and certain parameters and registers are associated with the current diversion level (the top non-diversion level may be thought of as the 0th diversion level). These are the diversion trap and associated macro, no-space mode, the internally-saved marked place (see `mk` and `rt`), the current vertical place (`.d` register), the current high-water text baseline (`.h` register), and the current diversion name (`.z` register).

7.5. Traps. Three types of trap mechanisms are available—page traps, a diversion trap, and an input-line-count trap. Macro-invocation traps may be planted using `wh` at any page position including the top. This trap position may be changed using `ch`. Trap positions at or below the bottom of the page have no effect unless or until moved to within the page or rendered effective by an increase in page length. Two traps may be planted at the same position only by first planting them at different positions and then moving one of the traps; the first planted trap will conceal the second unless and until the first one is moved (see Tutorial Examples). If the first one is moved back, it again conceals the second trap. The macro associated with a page trap is automatically invoked when a line of text is output whose vertical size reaches or sweeps past the trap position. Reaching the bottom of a page springs the top-of-page trap, if any, provided there is a next page. The distance to the next trap position is available in the `.t` register; if there are no traps between the current position and the bottom of the page, the distance returned is the distance to the page bottom.

A macro-invocation trap effective in the current diversion may be planted using `dt`. The `.t` register works in a diversion; if there is no subsequent trap a large distance is returned. For a description of input-line-count traps, see `it` below.

`.de` *xx yy* – *.yy=*`..` –

Define or redefine the macro *xx*. The contents of the macro begin on the next input line. Input lines are copied in *copy mode* until the definition is terminated by a line beginning with `.`*yy*, whereupon the macro *yy* is called. In the absence of *yy*, the definition is terminated by a line beginning with "`..`". A macro may contain `de` requests provided the terminating macros differ or the contained definition terminator is concealed. "`..`" can be concealed as `\\..` which will copy as `\..` and be reread as "`..`".

`.am` *xx yy* – *.yy=*`..` –

Append to macro *xx* (append version of `de`).

`.ds` *xx string* – ignored –

Define a string *xx* containing *string*. Any initial double quote in *string* is stripped off to permit initial blanks.

`.as` *xx string* – ignored –

Append *string* to string *xx* (append version of `ds`).

`.rm` *xx* – ignored –

Remove request, macro, or string. The name *xx* is removed from the name list and any related storage space is freed. Subsequent references will have no effect. If many macros and strings are being created dynamically, it may become necessary to remove unused ones to recapture internal storage space for newer registers.

`.rn` *xx yy* – ignored –

Rename request, macro, or string *xx* to *yy*. If *yy* exists, it is first removed.

`.di` *xx* – end D

Divert output to macro *xx*. Normal text processing occurs during diversion except that page offsetting is not done. The diversion ends when the request `di` or `da` is encountered without an argument; extraneous requests of this type should not appear when nested diversions are being used.

`.da` *xx* – end D

Divert, appending to macro *xx* (append version of `di`).

`.wh` *N xx* – – **v**

Install a trap to invoke *xx* at page position *N*; a negative N will be interpreted as a distance from the page bottom. Any macro previously planted at *N* is replaced by *xx*. A zero *N* refers to the top of a page. In the absence of *xx*, the first trap found at *N*, if any, is removed.

`.ch` *xx N* – – **v**

Change the trap position for macro *xx* to be *N*. In the absence of *N*, the trap, if any, is removed.

`.dt` *N xx* – off D,**v**

Install a diversion trap at position *N* in the *current* diversion to invoke macro *xx*. Another `dt` will redefine the diversion trap. If no arguments are given, the diversion trap is removed.

`.it` *N xx* – off E

Set an input-line-count trap to invoke the macro *xx* after *N* lines of *text* input have been read (control or request lines do not count). The text may be inline text or text interpolated by inline or trap-invoked macros.

`.em` *xx* none none –

The macro *xx* will be invoked when all input has ended. The effect is almost as if the contents of *xx* had been at the end of the last file processed, but all processing ceases at the next page eject.

8. Number Registers

A variety of parameters are available to the user as predefined *number registers* (see Summary, page 7). In addition, users may define their own registers. Register names are one or two characters long and do not conflict with request, macro, or string names. Except for certain predefined read-only registers, a number register can be read, written, automatically incremented or decremented, and interpolated into the input in a variety of formats. One common use of user-defined registers is to automatically number sections, paragraphs, lines, etc. A number register may be used any time numerical input is expected or desired and may be used in numerical *expressions* (§1.4).

Number registers are created and modified using `nr`, which specifies the name, numerical value, and the auto-increment size. Registers are also modified, if accessed with an auto-incrementing sequence. If the registers *x* and *xx* both contain *N* and have the auto-increment size *M*, the following access sequences have the effect shown:

Sequence	Effect on Register	Value Interpolated
`\n`*x*	none	*N*
`\n(`*xx*	none	*N*
`\n+`*x*	*x* incremented by *M*	*N+M*
`\n−`*x*	*x* decremented by *M*	*N−M*
`\n+(`*xx*	*xx* incremented by *M*	*N+M*
`\n−(`*xx*	*xx* decremented by *M*	*N−M*

When interpolated, a number register is converted to decimal (default), decimal with leading zeros, lower-case Roman, upper-case Roman, lower-case sequential alphabetic, or upper-case sequential alphabetic according to the format specified by `af`.

`.nr` *R* ±*N M* – **u**

The number register *R* is assigned the value ±*N* with respect to the previous value, if any. The increment for auto-incrementing is set to *M*.

`.af` *R c* arabic – –

Assign format *c* to register *R*. The available formats are:

Format	Numbering Sequence
`1`	0, 1, 2, 3, 4, 5, ...
`001`	000, 001, 002, 003, 004, 005, ...
`i`	0, i, ii, iii, iv, v, ...
`I`	0, I, II, III, IV, V, ...
`a`	0, a, b, c, ..., z, aa, ab, ..., zz, aaa, ...
`A`	0, A, B, C, ..., Z, AA, AB, ..., ZZ, AAA, ...

An arabic format having *N* digits specifies a field width of *N* digits (example 2 above). The read-only registers and the width function `\w` (§11.2) are always arabic. Warning: the value of a number register in a non-Arabic format is not numeric, and will not produce the expected results in expressions.

The function `\g`*x* or `\g(`*xx* returns the format of a number register in a form suitable for `af`; it returns nothing if the register has not been used.

`.rr` *R* – ignored –

Remove number register *R*. If many registers are being created dynamically, it may become necessary to remove unused registers to recapture internal storage space for newer registers. The register `.R` contains the number of number registers still available.

9. Tabs, Leaders, and Fields

9.1. Tabs and leaders. The ASCII horizontal tab character and the ASCII SOH (control-A, hereafter called the *leader* character) can both be used to generate either horizontal motion or a string of repeated characters. The length of the generated entity is governed by internal *tab stops* specifiable with `ta`. The default difference is that tabs generate motion and leaders generate a string of periods; `tc` and `lc` offer the choice of repeated character or motion. There are three types of internal tab stops—*left* adjusting, *right* adjusting, and *centering*. In the following table, *D* is the distance from the current position on the *input* line (where a tab or leader was found) to the next tab stop, *next−string* consists of the input characters following the tab (or leader) up to the next tab (or leader) or end of line, and *W* is the width of *next−string*.

Tab type	Length of motion or repeated characters	Location of *next−string*
Left	*D*	Following *D*
Right	*D−W*	Right adjusted within *D*
Centered	*D−W/*2	Centered on right end of *D*

The length of generated motion is allowed to be negative, but that of a repeated character string cannot be. Repeated character strings contain an integer number of characters, and any residual distance is prepended as motion. Tabs or leaders found after the last tab stop are ignored, but may be used as *next−string* terminators.

Tabs and leaders are not interpreted in copy mode. `\t` and `\a` always generate a non-interpreted tab and leader respectively, and are equivalent to actual tabs and leaders in copy mode.

9.2. Fields. A *field* is contained between a pair of *field delimiter* characters, and consists of substrings separated by *padding* indicator characters. The field length is the distance on the *input* line from the position where the field begins to the next tab stop. The difference between the total length of all the substrings and the field length is incorporated as horizontal padding space that is divided among the indicated padding places. The incorporated padding is allowed to be negative. For example, if the field delimiter is # and the padding indicator is ^, #^*xxx*^*right*# specifies a right-adjusted string with the string *xxx* centered in the remaining space.

`.ta` *Nt ...*	0.8; 0.5in	none	E,**m**

Set tab stops and types. *t*=R, right adjusting; *t*=C, centering; *t* absent, left adjusting. *Troff* tab stops are preset every 0.5in., *nroff* every 0.8in. The stop values are separated by spaces, and a value preceded by + is treated as an increment to the previous stop value.

`.tc` *c*	none	none	E

The tab repetition character becomes *c*, or is removed, thus specifying motion.

`.lc` *c*	.	none	E

The leader repetition character becomes *c*, or is removed, thus specifying motion.

`.fc` *a b*	off	off	–

The field delimiter is set to *a*; the padding indicator is set to the space character or to *b*, if given. In the absence of arguments the field mechanism is turned off.

10. Input and Output Conventions and Character Translations

10.1. Input character translations. Ways of inputting the valid character set were discussed in §2.1. The ASCII control characters horizontal tab (§9.1), SOH (§9.1), and backspace (§10.3) are discussed elsewhere. The newline delimits input lines. In addition, STX, ETX, ENQ, ACK, and BEL are accepted, and may be used as delimiters or translated into a graphic with `tr` (§10.5). All others are ignored.

The *escape* character \ introduces *escape sequences*, which cause the following character to mean another character, or to indicate some function. A complete list of such sequences is given in the Summary on page 7. The escape character \ should not be confused with the ASCII control character ESC. The escape character \ can be input with the sequence \\. The escape character can be changed with `ec`, and all that has been said about the default \ becomes true for the new escape character. `\e` can be used to print whatever the current escape character is. The escape mechanism may be turned off with `eo`, and restored with `ec`.

`.ec` *c*	\	\	–

Set escape character to \, or to *c*, if given.

`.eo`	on	–	–

Turn escape mechanism off.

10.2. Ligatures. The set of available ligatures is device and font dependent, but is often a subset of **fi**, **fl**, **ff**, **ffi**, and **ffl**. They may be input by `\(fi`, `\(fl`, `\(ff`, `\(Fi`, and `\(Fl` respectively. The ligature mode is normally on in *troff*, and automatically invokes ligatures during input.

`.lg` *N*	on; off	on	–

Ligature mode is turned on if *N* is absent or non-zero, and turned off if *N*=0. If *N*=2, only the two-character ligatures are automatically invoked. Ligature mode is inhibited for request, macro, string, register, or file names, and in copy mode. No effect in *nroff*.

10.3. Backspacing, underlining, overstriking, etc. Unless in copy mode, the ASCII backspace character is replaced by a backward horizontal motion having the width of the space character. Underlining as a form of line-drawing is discussed in §12.4. A generalized overstriking function is described in §12.1.

Nroff automatically underlines characters in the *underline* font, specifiable with `uf`, normally that on font position 2. In addition to `ft` and `\f`*F*, the underline font may be selected by `ul` and `cu`. Underlining is restricted to an output-device-dependent subset of reasonable characters.

`.ul` *N*	off	*N*=1	E

Italicize in *troff* (underline in *nroff*) the next *N* input text lines. Actually, switch to underline font, saving the current font for later restoration; other font changes within the span of a `ul` will take effect, but the restoration will undo the last change. Output generated by `tl` (§14) is affected by the font change, but does not decrement *N*. If $N > 1$, there is the risk that a trap interpolated macro may provide text lines within the span; environment switching can prevent this.

`.cu` *N*	off	*N*=1	E

Continuous underline. A variant of `ul` that causes *every* character to be underlined in *nroff*. Identical to `ul` in *troff*.

`.uf` *F*	Italic	Italic	–

Underline font set to *F*. In *nroff*, *F* may not be on position 1.

10.4. Control characters. Both the control character `.` and the *no–break* control character `'` may be changed. Such a change must be compatible with the design of any macros used in the span of the change, and particularly of any trap-invoked macros.

`.cc` *c*	.	.	E

The basic control character is set to *c*, or reset to "`.`".

`.c2` *c*	'	'	E

The *no–break* control character is set to *c*, or reset to "`'`".

10.5. Output translation. One character can be made a stand-in for another character using `tr`. All text processing (e.g., character comparisons) takes place with the input (stand-in) character, which appears to have the width of the final character. The graphic translation occurs at the moment of output (including diversion).

`.tr` *abcd....*	none	–	O

Translate *a* into *b*, *c* into *d*, etc. If an odd number of characters is given, the last one will be mapped into the space character. To be consistent, a particular translation must stay in effect from *input* to *output* time.

10.6. Transparent throughput. An input line beginning with a `\!` is read in copy mode and *transparently* output (without the initial `\!`); the text processor is otherwise unaware of the line's presence. This mechanism may be used to pass control information to a post-processor or to embed control lines in a macro created by a diversion.

10.7. Transparent output The sequence `\X'`*anything*`'` copies *anything* to the output, as a device control function of the form `x X` *anything* (§22). Escape sequences in *anything* are processed.

10.8. Comments and concealed newlines. An uncomfortably long input line that must stay one line (e.g., a string definition, or nofilled text) can be split into several physical lines by ending all but the last one with the escape `\`. The sequence `\`*newline* is always ignored, except in a comment. Comments may be embedded at the end of any line by prefacing them with `\"`. The newline at the end of a comment cannot be concealed. A line beginning with `\"` will appear as a blank line and behave like `.sp 1`; a comment can be on a line by itself by beginning the line with `.\"`.

11. Local Horizontal and Vertical Motions, and the Width Function

11.1. Local Motions. The functions `\v'`*N*`'` and `\h'`*N*`'` can be used for *local* vertical and horizontal motion respectively. The distance *N* may be negative; the positive directions are rightward and downward. A local motion is one contained within a line. To avoid unexpected vertical dislocations, it is necessary that the net vertical local motion within a word in filled text and otherwise within a line balance to zero. The escape sequences providing local motion are summarized in the following table.

<table>
<tr><th rowspan="2">Vertical
Local Motion</th><th colspan="2">Effect in</th><th rowspan="2">Horizontal
Local Motion</th><th colspan="2">Effect in</th></tr>
<tr><th>troff</th><th>nroff</th><th>troff</th><th>nroff</th></tr>
<tr><td>\v'N'
\h'N'</td><td colspan="2">Move distance N
Move distance N</td><td rowspan="2">\space
\0</td><td colspan="2" rowspan="2">Unpaddable space-size space
Digit-size space</td></tr>
<tr><td rowspan="2">\u
\d
\r</td><td rowspan="2">½ em up
½ em down
1 em up</td><td rowspan="2">½ line up
½ line down
1 line up</td></tr>
<tr><td>\|
\^</td><td>1/6 em space
1/12 em space</td><td>ignored
ignored</td></tr>
</table>

As an example, E^{2} could be generated by a sequence of size changes and motions: `E\s−2\v'−0.4m'2\v'0.4m'\s+2`; note that the 0.4 em vertical motions are at the smaller size.

11.2. Width Function. The *width* function `\w'`*string*`'` generates the numerical width of *string* (in basic units). Size and font changes may be embedded in *string*, and will not affect the current environment. For example, `.ti −\w'\fB1. 'u` could be used to temporarily indent leftward a distance equal to the size of the string "`1. `" in font B.

The width function also sets three number registers. The registers `st` and `sb` are set respectively to the highest and lowest extent of *string* relative to the baseline; then, for example, the total height of the string is `\n(stu−\n(sbu`. In *troff* the number register `ct` is set to a value between 0 and 3. The value 0 means that all of the characters in *string* were short lower case characters without descenders (like e); 1 means that at least one character has a descender (like y); 2 means that at least one character is tall (like H); and 3 means that both tall characters and characters with descenders are present.

11.3. Mark horizontal place. The function `\k`*x* causes the current horizontal position in the *input line* to be stored in register *x*. For example, the construction `\kx`*word*`\h'|\nxu+3u'`*word* will embolden *word* by backing up to almost its beginning and overprinting it, resulting in ***word***.

12. Overstrike, Bracket, Line−drawing, Graphics, and Zero−width Functions

12.1. Overstriking. Automatically centered overstriking of up to nine characters is provided by the *overstrike* function `\o'`*string*`'`. The characters in *string* are overprinted with centers aligned; the total width is that of the widest character. *string* may not contain local vertical motion. As examples, `\o'e\''` produces é, and `\o'\(mo\(sl'` produces ∉.

12.2. Zero−width characters. The function `\z`*c* will output *c* without spacing over it, and can be used to produce left−aligned overstruck combinations. As examples, `\z□+` will produce ⊞, and `\(br\z\(rn\(ul\(br` will produce a small badly constructed box |‾_|.

12.3. Large Brackets. The Special Font usually contains a number of bracket construction pieces ⎡ ⎣ ⎤ ⎦ ⎨ ⎬ ⎪ ⌊ ⌋ ⌈ ⌉ that can be combined into various bracket styles. The function `\b'`*string*`'` may be used to pile up vertically the characters in *string* (the first character on top and the last at the bottom); the characters are vertically separated by 1 em and the total pile is centered 1/2 em above the current baseline (½ line in *nroff*). For example,

```
\b'\(lc\(lf'E\b'\(rc\(rf'\x'-0.5m'\x'0.5m'
```

produces ⎡E⎤.

12.4. Line drawing. The function `\l'`*Nc*`'` (backslash-ell) draws a string of repeated *c*'s towards the right for a distance *N*. If *c* looks like a continuation of an expression for *N*, it may be insulated from *N* with \&. If *c* is not specified, the _ (baseline rule) is used (underline character in *nroff*). If *N* is negative, a backward horizontal motion of size *N* is made before drawing the string. Any space resulting from *N*/(size of *c*) having a remainder is put at the beginning (left end) of the string. If *N* is less than the width of *c*, a single *c* is centered on a distance *N*. In the case of characters that are designed to be connected, such as baseline-rule _, under-rule _, and root-en ‾, the remainder space is covered by overlapping. As an example, a macro to underscore a string can be written

```
.de us
\\$1\l'|0\(ul'
..
```

or one to draw a box around a string

```
.de bx
\(br\|\\$1\|\(br\l'|0\(rn'\l'|0\(ul'
..
```

such that

```
.ul "underlined words"
```

and

```
.bx "words in a box"
```

yield underlined words and |words in a box|.

The function `\L'`*Nc*`'` draws a vertical line consisting of the (optional) character *c* stacked vertically apart 1 em (1 line in *nroff*), with the first two characters overlapped, if necessary, to form a continuous line. The default character is the *box rule* | (`\(br`); the other suitable character is the *bold vertical* | (`\(bv`). The line is begun without any initial motion relative to the current baseline. A positive *N* specifies a line drawn downward and a negative *N* specifies a line drawn upward. After the line is drawn no compensating motions are made; the instantaneous baseline is at the end of the line.

The horizontal and vertical line drawing functions may be used in combination to produce large boxes. The zero-width *box-rule* and the ½-em wide *under-rule* were designed to form corners when using 1-em vertical spacings. For example the macro

```
.de eb
.sp -1   \"compensate for next automatic baseline spacing
.nf      \"avoid possibly overflowing word buffer
\h'-.5n'\L'|\\nau-1'\l'\\n(.lu+1n\(ul'\L'-|\\nau+1'\l'|0u-.5n\(ul'
.fi
..
```

will draw a box around some text whose beginning vertical place was saved in number register *a* (e.g., using `.mk a`) as was done for this paragraph.

12.5. Graphics. The function `\D'`*c...*`'` draws a graphic object of type *c* according to a sequence of parameters, which are generally pairs of numbers.

`\D'l` *dh dv*`'`	draw line from current position by *dh*, *dv*
`\D'c` *d*`'`	draw circle of diameter *d* with left side at current position
`\D'e` d_1 d_2`'`	draw ellipse of diameters d_1 and d_2

`\D'a` dh_1 dv_1 dh_2 dv_2`'` draw arc from current position to $dh_1 + dh_2$, $dv_1 + dv_2$, with center at dh_1, dv_1 from current position

`\D'~` $dh_1\, dv_1\, dh_2\, dv_2$ ...`'` draw B-spline from current position by dh_1, dv_1, then by dh_2, dv_2, then by dh_2, dv_2, then ...

For example, `\D'e0.2i 0.1i'` draws the ellipse , and `\D'l.2i −.1i'\D'l.1i .1i'` the line . A `\D` with an unknown *c* is processed and copied through to the output for unspecified interpretation; coordinates are interpreted alternately as horizontal and vertical values.

Numbers taken as horizontal (first, third, etc.) have default scaling of ems; vertical numbers (second, fourth, etc.) have default scaling of *V*s (§1.3). The position after a graphical object has been drawn is at its end; for circles and ellipses, the ''end'' is at the right side.

13. Hyphenation.

Automatic hyphenation may be switched off and on. When switched on with `hy`, several variants may be set. A *hyphenation indicator* character may be embedded in a word to specify desired hyphenation points, or may be prefixed to suppress hyphenation. In addition, the user may specify a small list of exception words.

Only words that consist of a central alphabetic string surrounded by (usually null) non-alphabetic strings are candidates for automatic hyphenation. Words that contain hyphens (minus), em-dashes (`\(em`), or hyphenation indicator characters are always subject to splitting after those characters, whether automatic hyphenation is on or off.

`.nh` hyphenate – E

Automatic hyphenation is turned off.

`.hy` *N* on, $N=1$ on, $N=1$ E

Automatic hyphenation is turned on for $N \geqq 1$, or off for $N=0$. If $N=2$, last lines (ones that will cause a trap) are not hyphenated. For $N=4$ and 8, the last and first two characters respectively of a word are not split off. These values are additive; i.e., $N=14$ will invoke all three restrictions.

`.hc` *c* \% \% E

Hyphenation indicator character is set to *c* or to the default \%. The indicator does not appear in the output.

`.hw` *word ...* ignored –

Specify hyphenation points in words with embedded minus signs. Versions of a word with terminal *s* are implied; i.e., `dig−it` implies `dig−its`. This list is examined initially and after each suffix stripping. The space available is small.

14. Three−Part Titles.

The titling function `tl` provides for automatic placement of three fields at the left, center, and right of a line with a title length specifiable with `lt`. `tl` may be used anywhere, and is independent of the normal text collecting process. A common use is in header and footer macros.

`.tl` '*left*'*center*'*right*' – –

The strings *left*, *center*, and *right* are respectively left-adjusted, centered, and right-adjusted in the current title length. Any of the strings may be empty, and overlapping is permitted. If the page-number character (initially %) is found within any of the fields it is replaced by the current page number in the format assigned to register %. Any character may be used in place of ' as the string delimiter.

`.pc` *c* % off –

The page number character is set to *c*, or removed. The page number register remains %.

`.lt` ±*N* 6.5 in previous E,**m**

Length of title is set to ±*N*. The line length and the title length are independent. Indents do not apply to titles; page offsets do.

15. Output Line Numbering.

Automatic sequence numbering of output lines may be requested with `nm`. When in effect, a three-digit, arabic number plus a digit-space is prefixed to output text lines. The text lines are thus offset by four digit-spaces, and otherwise retain their line length; a reduction in line length may be desired to keep the right margin aligned with an earlier margin. Blank lines, other vertical spaces, and lines generated by `tl` are not numbered. Numbering can be temporarily suspended with `nn`, or with an `.nm` followed by a later `.nm +0`. In addition, a line number indent *I*, and the number-text separation *S* may be specified in digit-spaces. Further, it can be specified that only those line numbers that are multiples of some number *M* are to be printed (the others will appear as blank number fields).

`.nm` ±*N M S I* off E

Line number mode. If ±*N* is given, line numbering is turned on, and the next output line numbered is numbered ±*N*. Default values are *M*=1, *S*=1, and *I*=0. Parameters corresponding to missing arguments are unaffected; a non-numeric argument is considered missing. In the absence of all arguments, numbering is turned off; the next line number is preserved for possible further use in number register `ln`.

`.nn` *N* – *N*=1 E

The next *N* text output lines are not numbered.

As an example, the paragraph portions of this section are numbered with *M*=3: `.nm 1 3` was placed at the beginning; `.nm` was placed at the end of the first paragraph; and `.nm +0` was placed in front of this paragraph; and `.nm` finally placed at the end. Line lengths were also changed (by `\w'0000'u`) to keep the right side aligned. Another example is `.nm +5 5 x 3`, which turns on numbering with the line number of the next line to be 5 greater than the last numbered line, with *M*=5, with spacing *S* untouched, and with the indent *I* set to 3.

16. Conditional Acceptance of Input

In the following, *c* is a one-character built-in *condition* name, `!` signifies *not*, *N* is a numerical expression, *string1* and *string2* are strings delimited by any non-blank, non-numeric character not in the strings, and *anything* represents what is conditionally accepted.

`.if` *c anything* – –

If condition *c* true, accept *anything* as input; in multi-line case use \{*anything*\}.

`.if !`*c anything* – –

If condition *c* false, accept *anything*.

`.if` *N anything* – **u**

If expression $N > 0$, accept *anything*.

`.if !`*N anything* – **u**

If expression $N \leq 0$ [sic], accept *anything*.

`.if 'string1'string2' anything` –

If *string1* identical to *string2*, accept *anything*.

`.if !'string1'string2' anything` –

If *string1* not identical to *string2*, accept *anything*.

`.ie c anything` – **u**

If portion of if-else; all of the forms for `if` above are valid.

`.el anything` – –

Else portion of if-else.

The built-in condition names are:

Condition Name	True If
`o`	Current page number is odd
`e`	Current page number is even
`t`	Formatter is *troff*
`n`	Formatter is *nroff*

If the condition *c* is true, or if the number *N* is greater than zero, or if the strings compare identically (including motions and character size and font), *anything* is accepted as input. If a ! precedes the condition, number, or string comparison, the sense of the acceptance is reversed.

Any spaces between the condition and the beginning of *anything* are skipped over. The *anything* can be either a single input line (text, macro, or whatever) or a number of input lines. In the multi-line case, the first line must begin with a left delimiter `\{` and the last line must end with a right delimiter `\}`.

The request `ie` (if-else) is identical to `if` except that the acceptance state is remembered. A subsequent and matching `el` (else) request then uses the reverse sense of that state. `ie-el` pairs may be nested.

Some examples are:

```
.if e .tl 'Even Page %'''
```

which outputs a title if the page number is even; and

```
.ie \n%>1 \{\
'       sp 0.5i
.       tl 'Page %'''
'       sp |1.2i \}
.el .sp |2.5i
```

which treats page 1 differently from other pages.

17. Environment Switching.

A number of the parameters that control the text processing are gathered together into an *environment*, which can be switched by the user. The environment parameters are those associated with requests noting E in their *Notes* column; in addition, partially collected lines and words are in the environment. Everything else is global; examples are page-oriented parameters, diversion-oriented parameters, number registers, and macro and string definitions. All environments are initialized with default parameter values.

`.ev N` $N=0$ previous –

Environment switched to environment $0 \leqq N \leqq 2$. Switching is done in push-down fashion so that restoring a previous environment *must* be done with

`.ev` rather than specific reference. Note that what is pushed down and restored is the environment *number,* not its contents.

18. Insertions from the Standard Input

The input can be temporarily switched to the system standard input with `rd`, which will switch back when two consecutive newlines are found (the extra blank line is not used). This mechanism is intended for insertions in form-letter-like documentation. The standard input can be the user's keyboard, a pipe, or a file.

`.rd` *prompt* – *prompt*=BEL –

Read insertion from the standard input until two newlines in a row are found. If the standard input is the user's keyboard, *prompt* (or a BEL) is written onto the standard output. `rd` behaves like a macro, and arguments may be placed after *prompt*.

`.ex` – – –

Exit from *nroff*/*troff*. Text processing is terminated exactly as if all input had ended.

If insertions are to be taken from the terminal keyboard while output is being printed on the terminal, the command line option `−q` will turn off the echoing of keyboard input and prompt only with BEL. The regular input and insertion input cannot simultaneously come from the standard input.

As an example, multiple copies of a form letter may be prepared by entering the insertions for all the copies in one file to be used as the standard input, and causing the file containing the letter to reinvoke itself with `nx` (§19); the process would ultimately be ended by an `ex` in the insertion file.

19. Input/Output File Switching

`.so` *filename* – –

Switch source file. The top input (file reading) level is switched to *filename*. When the new file ends, input is again taken from the original file. `so`'s may be nested.

`.nx` *filename* end-of-file –

Next file is *filename*. The current file is considered ended, and the input is immediately switched to *filename*.

`.sy` *string* – –

Execute program from *string*, which is the rest of the input line. The output is not collected automatically. The number register `$$`, which contains the process id of the *troff* process, may be useful in generating unique filenames for output.

`.pi` *string* – –

Pipe output to *string*, which is the rest of the input line. This request must occur before any printing occurs; typically it is the first line of input.

`.cf` *filename* – –

Copy contents of file *filename* to output, completely unprocessed. The file is assumed to contain something meaningful to subsequent processes.

20. Miscellaneous

`.mc` *c N*	–	off	E,**m**

Specifies that a *margin* character *c* appear a distance *N* to the right of the right margin after each non–empty text line (except those produced by `tl`). If the output line is too long (as can happen in nofill mode) the character will be appended to the line. If *N* is not given, the previous *N* is used; the initial *N* is 0.2 inches in *nroff* and 1 em in *troff*. The margin character used with this paragraph was a 12–point box–rule.

`.tm` *string*	–	newline	–

After skipping initial blanks, *string* (rest of the line) is read in copy mode and written on the standard error.

`.ab` *string*	–	newline	–

After skipping initial blanks, *string* (rest of the line) is read in copy mode and written on the standard error. *Troff* or *nroff* then exit.

`.ig` *yy*	–	*.yy*=`..`	–

Ignore input lines. `ig` behaves exactly like `de` (§7) except that the input is discarded. The input is read in copy mode, and any auto–incremented registers will be affected.

`.lf` *N filename*		–	–

Set line number to *N* and filename to *filename* for purposes of subsequent error messages, etc. The number register [sic] `.F` contains the name of the current input file, as set by command line argument, `so`, `nx`, or `lf`. The number register `.c` contains the number of input lines read from the current file, again perhaps as modified by `lf`.

`.pm` *t*	–	all	–

Print macros. The names and sizes of all of the defined macros and strings are printed on the standard error; if *t* is given, only the total of the sizes is printed. The sizes is given in blocks of 128 characters.

`.fl`	–	–	B

Flush output buffer. Force output, including any pending position information.

21. Output and Error Messages.

The output from `tm`, `pm`, and the prompt from `rd`, as well as various error messages, are written onto the standard error. The latter is different from the standard output, where formatted text goes. By default, both are written onto the user's terminal, but they can be independently redirected.

Various error conditions may occur during the operation of *nroff* and *troff*. Certain less serious errors having only local impact do not cause processing to terminate. Two examples are *word overflow*, caused by a word that is too large to fit into the word buffer (in fill mode), and *line overflow*, caused by an output line that grew too large to fit in the line buffer. In both cases, a message is printed, the offending excess is discarded, and the affected word or line is marked at the point of truncation with a ∗ in *nroff* and a ☜ in *troff*. Processing continues if possible, on the grounds that output useful for debugging may be produced. If a serious error occurs, processing terminates, and a message is printed, along with a list of the macro names currently active. Examples of serious errors include the inability to create, read, or write files, and the exceeding of certain internal limits that make future output unlikely to be useful.

22. Output Language

Troff produces its output in a language that is independent of any specific output device, except that the numbers in it have been computed on the basis of the resolution of the device, and the sizes, fonts, and characters that that device can print. Nevertheless it is quite possible to interpret that output on a different device, within the latter's capabilities.

`s`*n*	set point size to *n*
`f`*n*	set font to *n*
`c`*c*	print character *c*
`C`*name*	print the character called *name*; terminate *name* by white space
`N`*n*	print character *n* on current font
`H`*n*	go to absolute horizontal position *n* ($n \geqq 0$)
`V`*n*	go to absolute vertical position *n* ($n \geqq 0$, down is positive)
`h`*n*	go *n* units horizontally; $n < 0$ is to the left
`v`*n*	go *n* units vertically; $n < 0$ is up
nnc	move right *nn*, then print UTF character *c*; *nn* must be exactly 2 digits
`p`*n*	new page *n* begins—set vertical position to 0
`n`*b a*	end of line (information only—no action); *b* = space before line, *a* = after
`w`	paddable word space (information only—no action)
`D`*c* ...\n	graphics function *c*; see below
`x` ...\n	device control functions; see below
# ...\n	comment

All position values are in units. Sequences that end in digits must be followed by a non-digit. Blanks, tabs and newlines may occur as separators in the input, and are mandatory to separate constructions that would otherwise be confused. Graphics functions, device control functions, and comments extend to the end of the line they occur on.

The device control and graphics commands are intended as open-ended families, to be expanded as needed. The graphics functions coincide directly with the `\D` sequences:

`Dl` *dh dv*	draw line from current position by *dh*, *dv*
`Dc` *d*	draw circle of diameter *d* with left side here
`De` dh_1 dv_2	draw ellipse of diameters dh_1 and dv_2
`Da` dh_1 dv_1 dh_2 dv_2	draw arc from current position to $dh_1 + dh_2$, $dv_1 + dv_2$, center at dh_1, dv_1 from current position
`D~` dh_1 dv_1 dh_2 dv_2 ...	draw B-spline from current position to dh_1, dv_1, then to dh_2, dv_2, then to ...
`D`*z* dh_1 dv_1 dh_2 dv_2 ...	for any other *z* is uninterpreted

In all of these, *dh*, *dv* is an increment on the current horizontal and vertical position, with down and right positive. All distances and dimensions are in units.

The device control functions begin with `x`, then a command, then other parameters.

`x T` *s*	name of typesetter is *s*
`x r` *n h v*	resolution is *n* units/inch; *h* = minimum horizontal motion, *v* = minimum vertical
`x i`	initialize
`x f` *n s*	mount font *s* on font position *n*
`x p`	pause—can restart
`x s`	stop—done forever
`x t`	generate trailer information, if any
`x H` *n*	set character height to *n*
`x S` *n*	set slant to *n*

`x X` *any* generated by the `\X` function
`x` *any* to be ignored if not recognized

Subcommands like "`i`" may be spelled out like "`init`".

The commands `x T`, `x r` ..., and `x i` must occur first; fonts must be mounted before they can be used; `x s` comes last. There are no other order requirements.

The following is the output from "`hello, world`" for a typical printer, as described in §23:

```
x T utf
x res 720 1 1
x init
V0
p1

x font 1 R
x font 2 I
x font 3 B
x font 4 BI
x font 5 CW
x font 6 H
x font 7 HB
x font 8 HX
x font 9 S1
x font 10 S

s10
f1
H0
s10
f1
V0
H720
V120
ch
50e44l28l28o50,w58w72o50r33l28dn120 0
x trailer
V7920
x stop
```

Troff output is normally not redundant; size and font changes and position information are not included unless needed. Nevertheless, each page is self-contained, for the benefit of postprocessors that re-order pages or process only a subset.

23. Device and Font Description Files

The parameters that describe a output device *name* are read from the directory `/sys/lib/troff/font/dev`*name*, each time *troff* is invoked. The device name is provided by default, by the environment variable TYPESETTER, or by a command-line argument `−T`*name*. The default device name is `utf`, for UTF-encoded Unicode characters. The pre-defined string `.T` contains the name of the device. The `−F` command-line option may be used to change the default directory.

23.1. Device description file. General parameters of the device are stored, one per line, in the file `/sys/lib/troff/font/dev`*name*`/DESC`, as a sequence of names and values. *Troff* recognizes these parameters, and ignores any others that may be present for specific drivers:

`fonts` n F_1 F_2 ... F_n
`sizes` s_1 s_2 ...0

```
res n
hor n
vert n
unitwidth n
charset
```
list of multi−character character names (optional)

The F_i are font names to be initially mounted. The list of sizes is a set of integers representing some or all of the legal sizes the device can produce, terminated by a zero. The `res` parameter gives the resolution of the machine in units per inch; `hor` and `ver` give the minimum number of units that can be moved horizontally and vertically.

Character widths for each font are assumed to be given in machine units at point size `unitwidth`. (In other words, a character with a width of *n* is *n* units wide at size `unitwidth`.) All widths are integers at all sizes.

A list of valid character names may be introduced by `charset`; the list of names is optional.

A line whose first non−blank character is # is a comment. Except that `charset` must occur last, parameters may appear in any order.

Here is a subset of the DESC file for a typical Postscript printer:

```
# Description file for Postscript printers.

fonts 10 R I B BI CW H HB HX S1 S
sizes 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
   24 25 26 27 28 29 30 31 32 33 34 35 36 38 40 44 48 54 60 72 0
res 720
hor 1
vert 1
unitwidth 10
charset
hy ct fi fl ff Fi Fl dg em 14 34 12 en aa
ga ru sc dd −> br Sl ps cs cy as os =. ld
rd le ge pp −+ ob vr
sq bx ci fa te ** pl mi eq ~= *A *B *X *D
*E *F *G *Y *I *K *L *M *N *O *P *R *H *S *T *U *W
*C *Q *Z ul rn *a *b *x *d *e *f *g *y *i *k
*l *m *n *o *p *h *r *s *t *u *w *c *q *z
```

23.2. Font description files. Each font is described by an analogous description file, which begins with parameters of the font, one per line, followed by a list of characters and widths. The file for font *f* is `/sys/lib/troff/font/dev`*name*`/`*f*.

`name` *str*	name of font is *str*
`ligatures ... 0`	list of ligatures
`spacewidth` *n*	width of a space on this font
`special`	this is a special font
`charset`	

list of character name, width, ascender/descender, code, tab separated

The `name` and `charset` fields are mandatory; `charset` must be last. Comments are permitted, as are other unrecognized parameters.

Each line following `charset` describes one character: its name, its width in units as described above, ascender/descender information, and a decimal, octal or hexadecimal value by which the output device knows it (the `\N` ‘‘number’’ of the character). The character name is arbitrary, except that −−− signifies an unnamed character. If the width field contains ", the name is a synonym for the previous character. The ascender/descender field is 1 if the character has a descender (hangs below the baseline, like y), is 2 if it has an ascender (is tall, like Y), is 3 if both, and is 0 if neither. The value is returned in the ct register, as computed by the \w function (§11.2).

Here are excerpts from a typical font description file for the same Postscript printer.

```
hy      33      0      45      hyphen \(hy
-       "                      - is a synonym for \(hy

Q       72      3      81

a       44      0      97
b       50      2      98
c       44      0      99
d       50      2      100
y       50      1      121

em      100     0      208
---     44      2      220     Pound symbol £, \N'220'
---     36      0      221     centered dot \N'221'
```

This says, for example, that the width of the letter a is 44 units at point size 10, the value of unitwidth. Point sizes are scaled linearly and rounded, so the width of a will be 44 at size 10, 40 at size 9, 35 at size 8, and so on.

Tutorial Examples

Introduction

It is almost always necessary to prepare at least a small set of macro definitions to describe a document. Such common formatting needs as page margins and footnotes are deliberately not built into *nroff* and *troff*. Instead, the macro and string definition, number register, diversion, environment switching, page-position trap, and conditional input mechanisms provide the basis for user-defined implementations.

For most uses, a standard package like −ms or −mm is the right choice. The next stage is to augment that, or to selectively replace macros from the standard package. The last stage, much harder, is to write one's own from scratch. This is not a task for the novice.

The examples discussed here are intended to be useful and somewhat realistic, but will not necessarily cover all relevant contingencies. Explicit numerical parameters are used in the examples to make them easier to read and to illustrate typical values. In many cases, number registers would be used to reduce the number of places where numerical information is kept, and to concentrate conditional parameter initialization like that which depends on whether *troff* or *nroff* is being used.

Page Margins

As discussed in §3, header and footer macros are usually defined to describe the top and bottom page margin areas respectively. A trap is planted at page position 0 for the header, and at −*N* (*N* from the page bottom) for the footer. The simplest such definitions might be

```
.de hd  \"define header
'sp 1i
..      \"end definition
.de fo  \"define footer
'bp
..      \"end definition
.wh 0 hd
.wh −1i fo
```

which provide blank 1 inch top and bottom margins. The header will occur on the *first* page only if the definition and trap exist prior to the initial pseudo-page transition (§3). In fill mode, the output line that springs the footer trap was typically forced out because some part or whole word didn't fit on it. If anything in the footer and header that follows causes a break, that word or part word will be forced out. In this and other examples, requests like bp and sp that normally cause breaks are invoked using the no-break control character ' to avoid this. When the header/footer design contains material requiring independent text processing, the environment may be switched, avoiding most interaction with the running text.

A more realistic example would be

```
.de hd  \"header
.if \\n%>1 \{\
'sp  0.5i−1     \"tl base at 0.5i
.tl ''− % −''   \"centered page number
.ps      \"restore size
.ft      \"restore font
.vs  \} \"restore vs
'sp  1.0i       \"space to 1.0i
.ns      \"turn on no−space mode
..
.de fo  \"footer
.ps 10  \"set footer/header size
.ft R   \"set font
.vs 12p \"set baseline spacing
.if \\n%=1 \{\
'sp  \\n(.pu−0.5i−1  \"tl base 0.5i up
.tl ''− % −'' \}  \"first page number
'bp
..
.wh 0 hd
.wh −1i fo
```

which sets the size, font, and baseline spacing for the header/footer material, and ultimately restores them. The material in this case is a page number at the bottom of the first page and at the top of the remaining pages. The `sp`'s refer to absolute positions to avoid dependence on the baseline spacing. Another reason for doing this in the footer is that the footer is invoked by printing a line whose vertical spacing swept past the trap position by possibly as much as the baseline spacing. No−space mode is turned on at the end of `hd` to render ineffective accidental occurrences of `sp` at the top of the running text.

This method of restoring size, font, etc., presupposes that such requests (that set *previous* value) are *not* used in the running text. A better scheme is to save and restore both the current *and* previous values as shown for size in the following:

```
.de fo
.nr s1 \\n(.s   \"current size
.ps
.nr s2 \\n(.s   \"previous size
.  −−−  \"rest of footer
..
.de hd
.  −−−  \"header stuff
.ps \\n(s2  \"restore previous size
.ps \\n(s1  \"restore current size
..
```

Page numbers may be printed in the bottom margin by a separate macro triggered during the footer's page ejection:

```
.de bn  \"bottom number
.tl ''− % −''   \"centered page number
..
.wh −0.5i−1v bn  \"tl base 0.5i up
```

Paragraphs and Headings

The housekeeping associated with starting a new paragraph should be collected in a paragraph macro that, for example, does the desired preparagraph spacing, forces the correct font, size, baseline spacing, and indent, checks that enough space remains for *more than one* line, and requests a temporary indent.

```
.de pg    \"paragraph
.br       \"break
.ft R     \"force font,
.ps 10    \"size,
.vs 12p   \"spacing,
.in 0     \"and indent
.sp 0.4   \"prespace
.ne 1+\\n(.Vu  \"want more than 1 line
.ti 0.2i        \"temp indent
..
```

The first break in pg will force out any previous partial lines, and must occur before the vs. The forcing of font, etc., is partly a defense against prior error and partly to permit things like section heading macros to set parameters only once. The prespacing parameter is suitable for *troff*; a larger space, at least as big as the output device vertical resolution, would be more suitable in *nroff*. The choice of remaining space to test for in the ne is the smallest amount greater than one line (the .V is the available vertical resolution).

A macro to automatically number section headings might look like:

```
.de sc  \"section
.  ---  \"force font, etc.
.sp 0.4 \"prespace
.ne 2.4+\\n(.Vu \"want 2.4+ lines
.fi
\\n+S.
..
.nr S 0 1       \"init S
```

The usage is .sc, followed by the section heading text, followed by .pg. The ne test value includes one line of heading, 0.4 line in the following pg, and one line of the paragraph text. A word consisting of the next section number and a period is produced to begin the heading line. The format of the number may be set by af (§8).

Another common form is the labeled, indented paragraph, where the label protrudes left into the indent space.

```
.de lp  \"labeled paragraph
.pg
.in 0.5i        \"paragraph indent
.ta 0.2i 0.5i   \"label, paragraph
.ti 0
\t\\$1\t\c      \"flow into paragraph
..
```

The intended usage is ".lp *label*"; *label* will begin at 0.2 inch, and cannot exceed a length of 0.3 inch without intruding into the paragraph. The label could be right adjusted against 0.4 inch by setting the tabs instead with .ta 0.4iR 0.5i. The last line of lp ends with \c so that it will become a part of the first line of the text that follows.

Multiple Column Output

The production of multiple column pages requires the footer macro to decide whether it was invoked by other than the last column, so that it will begin a new column rather than produce the bottom margin. The header can initialize a column register that the footer will increment and test. The following is arranged for two columns, but is easily modified for more.

```
.de hd  \"header
.  ---
.nr cl 0 1      \"init column count
.mk     \"mark top of text
..

.de fo  \"footer
.ie \\n+(cl<2 \{\
.po +3.4i       \"next column; 3.1+0.3
.rt     \"back to mark
.ns \}  \"no-space mode
.el \{\
.po \\nMu       \"restore left margin
.  ---
'bp \}
..
.ll 3.1i        \"column width
.nr M \\n(.o    \"save left margin
```

Typically a portion of the top of the first page contains full width text; the request for the narrower line length, as well as another `.mk` would be made where the two column output was to begin.

Footnotes

The footnote mechanism to be described is used by embedding the footnotes in the input text at the point of reference, demarcated by an initial `.fn` and a terminal `.ef`:

```
.fn
Footnote text and control lines...
.ef
```

In the following, footnotes are processed in a separate environment and diverted for later printing in the space immediately prior to the bottom margin. There is provision for the case where the last collected footnote doesn't completely fit in the available space.

```
.de hd  \"header
.  ---
.nr x 0 1       \"init footnote count
.nr y 0-\\nb    \"current footer place
.ch fo -\\nbu   \"reset footer trap
.if \\n(dn .fz  \"leftover footnote
..

.de fo  \"footer
.nr dn 0  \"zero last diversion size
.if \\nx \{\
.ev 1   \"expand footnotes in ev1
.nf     \"retain vertical size
.FN     \"footnotes
.rm FN  \"delete it

.if "\\n(.z"fy" .di  \"end overflow di
.nr x 0 \"disable fx
.ev  \} \"pop environment
.  ---
'bp
..
```

```
.de fx  \"process footnote overflow
.if \\nx .di fy \"divert overflow
..

.de fn  \"start footnote
.da FN  \"divert (append) footnote
.ev 1   \"in environment 1
.if \\n+x=1 .fs   \"if 1st, separator
.fi     \"fill mode
..

.de ef  \"end footnote
.br     \"finish output
.nr z \\n(.v    \"save spacing
.ev     \"pop ev
.di     \"end diversion
.nr y -\\n(dn   \"new footer position,
.if \\nx=1 .nr y -(\\n(.v-\\nz) \
        \"uncertainty correction
.ch fo \\nyu    \"y is negative
.if (\\n(nl+1v)>(\\n(.p+\\ny) \
.ch fo \\n(nlu+1v        \"didn't fit
..

.de fs  \"separator
\l'1i'  \"1 inch rule
.br
..

.de fz  \"get leftover footnote
.fn
.nf     \"retain vertical size
.fy     \"where fx put it
.ef
..

.nr b 1.0i  \"bottom margin size
.wh 0 hd    \"header trap
.wh 12i fo  \"footer trap->temp pos
.wh -\\nbu fx   \"fx at footer position
.ch fo -\\nbu   \"conceal fx with fo
```

The header hd initializes a footnote count register x, and sets both the current footer trap position register y and the footer trap itself to a nominal position specified in register b. In addition, if the register dn indicates a leftover footnote, fz is invoked to reprocess it. The footnote start macro fn begins a diversion (append) in environment 1, and increments the count x; if the count is one, the footnote separator fs is interpolated. The separator is kept in a separate macro to permit user redefinition.

The footnote end macro ef restores the previous environment and ends the diversion after saving the spacing size in register z. y is then decremented by the size of the footnote, available in dn; then on the first footnote, y is further decremented by the difference in vertical baseline spacings of the two environments, to prevent the late triggering of the footer trap from causing the last line of the combined footnotes to overflow. The footer trap is then set to the lower (on the page) of y or the current page position (nl) plus one line, to allow for printing the reference line.

If indicated by x, the footer fo rereads the footnotes from FN in nofill mode in environment 1, and deletes FN. If the footnotes were too large to fit, the macro fx will be trap-invoked to redivert the overflow into `fy`, and the register `dn` will later indicate to the header whether `fy` is empty.

Both `fo` and `fx` are planted in the nominal footer trap position in an order that causes `fx` to be concealed unless the `fo` trap is moved. The footer then terminates the overflow diversion, if necessary, and zeros `x` to disable `fx`, because the uncertainty correction together with a not-too-late triggering of the footer can result in the footnote rereading finishing before reaching the `fx` trap.

A good exercise for the student is to combine the multiple-column and footnote mechanisms.

The Last Page

After the last input file has ended, *nroff* and *troff* invoke the *end macro* (§7), if any, and when it finishes, eject the remainder of the page. During the eject, any traps encountered are processed normally. At the end of this last page, processing terminates unless a partial line, word, or partial word remains. If it is desired that another page be started, the end-macro

```
.de en  \"end-macro
\c
'bp
..
.em en
```

will deposit a null partial word, and produce another last page.

Special Character Names

The following table lists names for a set of characters, most of which have traditionally been provided by *troff* using the ‘special’ or ‘symbol’ font. Many of these sequences are old ways to get what are now Unicode characters; Lucida Sans, for example, has glyphs corresponding to many of these but does not have the special sequences. Therefore the *troff* sequence \(*F gives the character Φ from the Times font instead of the character Φ from the current font, in this case Lucida Sans. Not all sequences print on any particular device, including this one; Peter faces appear in their place.

´	\'	μ	\(*m	≃	\(|=
`	\`	ν	\(*n	~	\(ap
—	\(em	ξ	\(*c	≠	\(!=
–	\(en	ο	\(*o	→	\(->
‐	\(hy	π	\(*p	←	\(<-
−	\-	ρ	\(*r	↑	\(ua
•	\(bu	σ	\(*s	↓	\(da
□	\(sq	ς	\(ts	×	\(mu
_	\(ru	τ	\(*t	÷	\(di
	\(14	υ	\(*u	±	\(+-
	\(12	φ	\(*f	∪	\(cu
	\(34	χ	\(*x	∩	\(ca
	\(fi	ψ	\(*q	⊂	\(sb
	\(fl	ω	\(*w	⊃	\(sp
	\(ff	Α	\(*A	⊆	\(ib
	\(Fi	Β	\(*B	⊇	\(ip
	\(Fl	Γ	\(*G	∞	\(if
°	\(de	Δ	\(*D	∂	\(pd
†	\(dg	Ε	\(*E	∇	\(gr
′	\(fm	Ζ	\(*Z	¬	\(no
	\(ct	Η	\(*Y	∫	\(is
®	\(rg	Θ	\(*H	∝	\(pt
©	\(co	Ι	\(*I	∅	\(es
+	\(pl	Κ	\(*K	∈	\(mo
−	\(mi	Λ	\(*L	|	\(br
=	\(eq	Μ	\(*M	‡	\(dd
∗	\(**	Ν	\(*N	☞	\(rh
	\(sc	Ξ	\(*C	☜	\(lh
´	\(aa	Ο	\(*O		\(bs
`	\(ga	Π	\(*P	|	\(or
_	\(ul	Ρ	\(*R	○	\(ci
/	\(sl	Σ	\(*S	⎡	\(lt
α	\(*a	Τ	\(*T	⎣	\(lb
β	\(*b	Υ	\(*U	⎤	\(rt
γ	\(*g	Φ	\(*F	⎦	\(rb
δ	\(*d	Χ	\(*X	⎨	\(lk
ε	\(*e	Ψ	\(*Q	⎬	\(rk
ζ	\(*z	Ω	\(*W	⎪	\(bv
η	\(*y	√	\(sr	⌊	\(lf
θ	\(*h		\(rn	⌋	\(rf
ι	\(*i	≥	\(>=	⌈	\(lc
κ	\(*k	≤	\(<=	⌉	\(rc
λ	\(*l	≡	\(==		\(LH