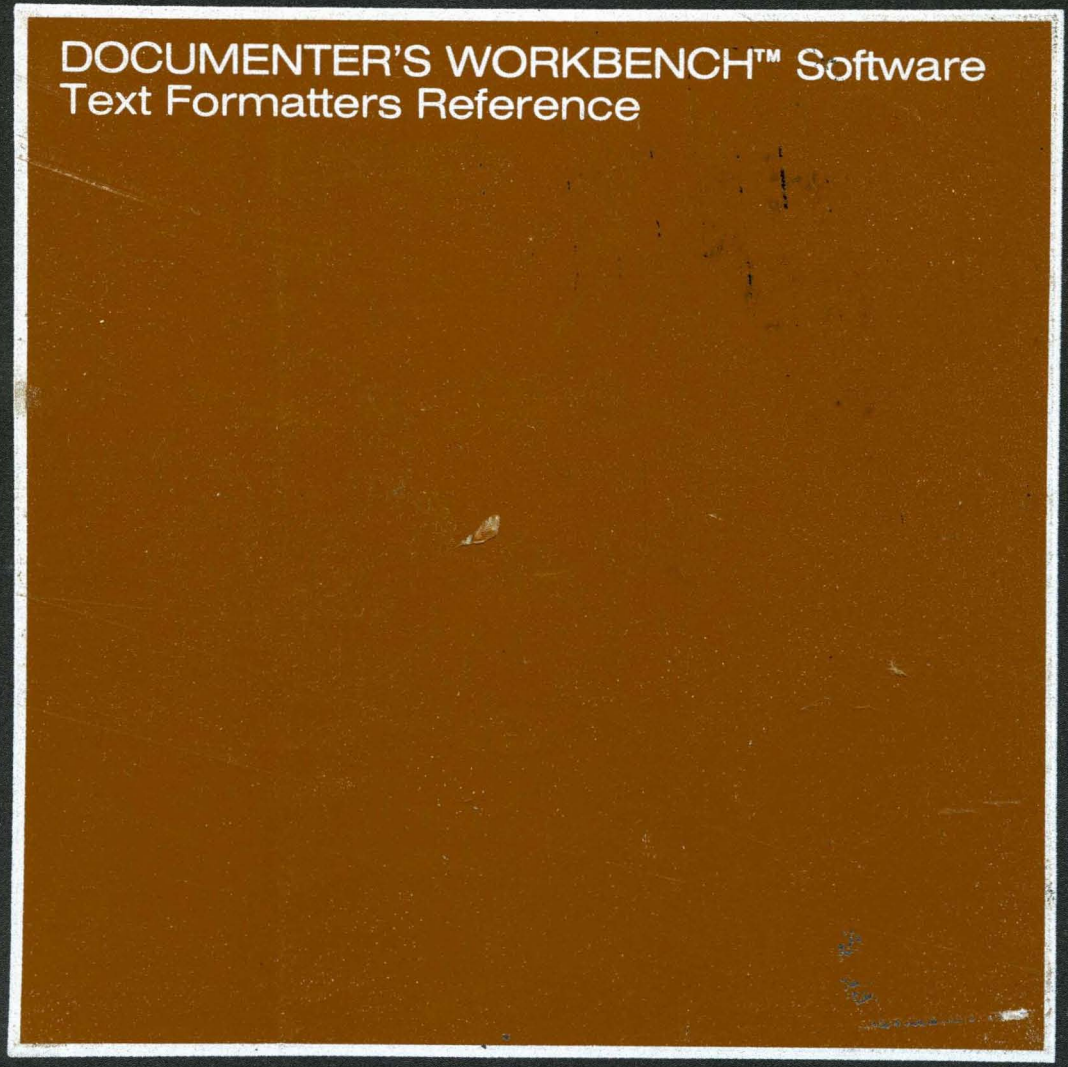
UNIX™ System V

DOCUMENTER'S WORKBENCH™ Software
Text Formatters Reference



Western Electric

December 1983
307-151 Issue 1

UNIX™ System V Release 2.0

DOCUMENTER'S WORKBENCH™ Software Text Formatters Reference

Western Electric

CONTENTS

Chapter 1	**INTRODUCTION**
Chapter 2	**NROFF/TROFF TUTORIAL**
Chapter 3	**NROFF AND TROFF USER MANUAL**
Chapter 4	**DEVICE-INDEPENDENT TROFF**
Chapter 5	**SROFF TUTORIAL GUIDE**
Chapter 6	**SROFF REFERENCE MANUAL**

Chapter 1

INTRODUCTION

PAGE

TEXT FORMATTERS 1-1

Chapter 1

INTRODUCTION

This book is a guide and reference manual for the text formatters that are provided with the UNIX* system DOCUMENTER'S WORKBENCH† software. This software provides an integrated set of text processing tools for easy, flexible, and professional documentation production. Books that describe other aspects of the DOCUMENTER'S WORKBENCH software are:

- *Introduction and Reference Manual*—Select Code 307-150
- *Macro Packages Reference*—Select Code 307-152
- *Preprocessors Reference*—Select Code 307-153.

The beginning user should refer to the DOCUMENTER'S WORKBENCH software *Introduction and Reference Manual* for a better overall description of the text processing tools available on the UNIX system.

TEXT FORMATTERS

On the DOCUMENTER'S WORKBENCH software, the text formatting programs provide control of text format by the use of requests (sometimes called formatter primitives) that are mixed in with the text to be formatted. These requests normally consist of two lowercase letters preceded by a period, on a line by themselves in the text file. The request may be followed on the same line by numbers or letters that provide the formatter with more information about the function of the request. The formatter requests provide a low-level control of text formatting for items such as indention, line length, spacing, filling, adjusting, centering, and titles.

---

* Trademark of Bell Laboratories.

† Trademark of Western Electric.

INTRODUCTION

The text formatters covered in this book are:

- Chapter 2 (*NROFF/TROFF TUTORIAL*)—Presents examples and explanations of formatting activities that you would use with **nroff** and **troff**.

- Chapter 3 (*NROFF/TROFF USER MANUAL*)—Provides a description of **nroff**, a formatter designed to produce output for simple typewriter-like terminals, **troff**, a formatter designed to produce output for phototypesetters that is phototypesetter independent, and **otroff**, a formatter designed to produce output for the Wang Laboratories CAT phototypesetter only. The language of these three formatters is very similar and in most cases they are compatible.

- Chapter 4 (*DEVICE INDEPENDENT TROFF*)—Provides a description of the **troff** formatter in which the differences between **troff** and **otroff** are pointed out.

- Chapter 5 (*SROFF TUTORIAL*)—Presents examples and explanations of the capablities of **sroff**.

- Chapter 6 (*SROFF USER MANUAL*)—Provides a description of the **sroff** formatter which is designed to produce output for simple printers. The **sroff** formatter is faster, but simpler, than the **nroff** text formatter.

Chapter 2

NROFF/TROFF TUTORIAL

PAGE

OVERVIEW 2-1

TUTORIAL TOPICS 2-3
1. Point Sizes and Line Spacing 2-3
2. Fonts and Special Characters 2-5
3. Indents and Line Lengths 2-7
4. Tabs 2-9
5. Local Motions 2-10
6. Strings 2-14
7. Introduction to Macros 2-15
8. Titles, Pages, and Page Numbering 2-17
9. Number Registers and Arithmetic 2-21
10. Macros With Arguments 2-24
11. Conditionals 2-27
12. Environments 2-29
13. Diversions 2-30

TUTORIAL EXAMPLES 2-32
1. Page Margins 2-32
2. Paragraphs and Headings 2-34
3. Multiple Column Output 2-36
4. Footnote Processing 2-37
5. Last Page 2-40

Chapter 2

NROFF/TROFF TUTORIAL

OVERVIEW

An important rule to remember when using the **troff** formatter is to use it through an intermediary. In many ways the **troff** formatter resembles an assembly language, remarkably powerful and flexible, but nonetheless such that many operations must be specified at a level of detail and in a form that is difficult to use.

There are programs that provide an interface to the **troff** formatter for the majority of users for three special applications.

- The **eqn** program provides an easy to learn language for typesetting mathematics. The user does not need to know the **troff** formatter to typeset mathematics.
- The **tbl** program provides an easy to learn language for producing tables of arbitrary complexity.
- The **pic** program provides an easy to learn language for typesetting graphics that includes several picture elements such as, boxes, circles, ellipses, arcs, lines, and arrows. It allows arbitrary positioning and sizing of picture elements and text.

These three programs are nroff/troff preprocessors. More information on the preprocessors can be found in the DOCUMENTER'S WORKBENCH Software *Preprocessors Reference*—Select code 307-153.

For producing general documents, there are a number of macro packages that define formatting rules and operations for specific styles of documents and reduce the amount of direct contact with the **troff** formatter. In particular, the Memorandum Macros (MM) package provides most of the facilities needed for a wide range of document preparation. There are also packages for viewgraphs and other special applications. These packages are easier to use than the

troff formatter language. They should be considered first. More information on the macro packages is in the *Macro Packages Reference*—select code 307-152.

In the few cases where existing packages do not accomplish the job, the solution is not to write an entirely new set of **troff** instructions from scratch but to make small changes to adapt packages that already exist. In accordance with this philosophy, the part of the **troff** formatter described here is only a small part of the whole, although it tries to concentrate on the more useful parts. The emphasis is on doing simple things and making incremental changes to what already exists.

To use the **troff** formatter, the actual text must be prepared plus some information that describes how it is to be printed. Text and formatting information are intimately intertwined. Most commands to the **troff** formatter are placed on a line separate from the text itself, one command per line beginning with a period. For example:

```
Some text.
.ps 14
Some more text.
```

will change the point size of the letters being printed to 14 point (one point is 1/72 of an inch).

Occasionally, something special occurs in the middle of a line, such as an exponent. The backslash (\) is used to introduce **troff** commands and special characters within a line of text.

TUTORIAL TOPICS

1. Point Sizes and Line Spacing

The **.ps** request sets the point size. Since one point is 1/72 inch, 6-point characters are 1/12 inch high, and 36-point characters are 1/2 inch high. There are 15 point sizes with the **otroff** formatter: 6, 7, 8, 9, 10, 11, 12, 14, 16, 18, 20, 22, 24, 28, and 36. Available point sizes with the **troff** formatter depend on the typesetter. Point size is rounded to the closest valid value, if the number following the **.ps** request is not a legal value.

If no number follows the **.ps** request, point size reverts to the previous value. The **troff** and **otroff** processors begin with point size 10. Point size can also be changed in the middle of a line or a word with a **\s** escape sequence. The **\s** sequence should be followed by a legal point size, except that the **\s0** sequence causes the size to revert to its previous value. The **\s1011** sequence is understood correctly as "point size 10, followed by an 11".

Relative size changes are also legal and useful:

```
\s-2UNCLE\s+2
```

temporarily decreases the size by two points, then restores it. Relative size changes have the advantage that the size difference is independent of the starting size of the document. The amount of the relative change is restricted to a single digit.

Another parameter that determines what the type looks like is the spacing between lines. It is set independently of the point size. Vertical spacing is measured from the bottom of one line to the bottom of the next. The command to control vertical spacing is **.vs**. For running text, it is usually best to set the vertical spacing about 20 percent larger than the character size. For example, a usable combination would be

```
.ps 9
.vs 11p
```

Vertical spacing is partly a matter of taste, depending on how much text is to be squeezed into a given space, and partly a matter of traditional printing style. By default, the **troff** and **otroff** formatters use a point size of 10 and a vertical spacing of 12. When **.vs** is used without arguments, vertical spacing reverts to the previous value.

The **.sp** request is used to get extra vertical space. Used alone, it gives one extra blank line (at whatever value **.vs** is set). Since that may be more or less than desired, **.sp** can be followed by information about how much space is wanted. For instance:

.sp 1.5i	means "a space of 1.5 inches" (most **troff** processor installations understand decimal fractions)
.sp 2i	means "two inches of vertical space"
.sp 2p	means "two points of vertical space"
.sp 2 or .sp 2v	means "two vertical spaces" (two of whatever **.vs** is set).

These same scale factors can be used after the **.vs** request to define line spacing. Scale factors can be used after most commands that deal with physical dimensions.

All size numbers are converted internally to *machine units*, which, for the Wang C/A/T phototypesetter and **otroff**, are 1/432 inch (1/6 point). For most purposes, this is enough resolution to provide good accuracy of representation. The situation is not quite so good vertically, where resolution is 1/144 inch (1/2 point). With the **troff** formatter, the resolution is typesetter dependent. The APS-5 typesetter has a resolution of 723 units per inch.

2. Fonts and Special Characters

The **otroff** processor and the Wang C/A/T phototypesetter allows four different fonts at one time. Normally, three fonts (Times Roman, Times Italic, and Times Bold) and one collection of special characters are permanently mounted.

With the **troff** formatter, available fonts and names are dependent upon the typesetter. Refer to Chapters 3 and 4 of this book for a more complete description of fonts, point sizes, and differences between the **otroff** and **troff** formatters.

The **otroff** processor prints in Roman unless otherwise commanded. To change the font, the **.ft** request is used:

.ft B	switch to bold font.
.ft I	switch to italics font.
.ft R	switch to Roman font.
.ft P	return to previous font.
.ft	return to previous font.

The underline request (**.ul**) causes the next input line to print in italics. It can be followed by a number to indicate that more than one line is to be italicized.

Fonts can be changed within a line or word with the **\f** in-line sequences. For instance:

bold*face* text

is produced by

```
\fBbold\fIface\fR text
```

If it is desired to do this so the previous font is left undisturbed, extra **\fP** sequences should be inserted:

```
\fBbold\fP\fIface\fP\fR text\fP
```

Since only the immediately previous font is remembered, the previous font must be restored after each change or it will be lost. The same is true of **.ps** and **.vs** when used without an argument.

There are other fonts available besides the standard set. The **.fp** request tells the **troff** formatter which fonts are actually mounted on the typesetter. For example:

```
.fp 3 H
```

says that the Helvetica font is mounted on position 3. Appropriate **.fp** requests should appear at the beginning of a document if standard fonts are not used.

It is possible to make a document relatively independent of the actual fonts used to print it by using font numbers instead of names. For example: **\f3** and **.ft3** mean "whatever font is mounted at position 3". Normal settings are Roman font on 1, italic on 2, bold on 3, and special on 4 (**otroff**).

There is also a way to get synthetic bold fonts by overstriking letters with a slight offset. The **.bd** request addresses this function.

Special characters have 4-character input names beginning with \(and may be inserted anywhere in the text. In particular, Greek letters are all of the form **\(*** –, where – is an uppercase or lowercase Roman font letter reminiscent of the Greek. A list of these special names is given in Chapter 3, Figure 3-5.

Some characters are automatically translated into others: grave and acute accents become open and close single quotation marks. Similarly, a typed minus sign becomes a hyphen. The \– input will print an explicit minus sign. A **\e** entry causes a backslash to be printed.

3. Indents and Line Lengths

The **troff** processor starts with a line length of 6.5 inches, which is too wide for 8-1/2 inch by 11-inch paper. The **.ll** request resets the line length. For example:

```
.ll 6i
```

As with the **.sp** request, the actual length can be specified in several ways; inches are probably the most intuitive. The maximum line length provided by the C/A/T phototypesetter and **otroff** is 7.54 inches. Again, this may be different when using **troff** with another phototypesetter. To use the full width, the default physical left margin (page offset) must be reset. This is done by the **.po** request. The margin is normally slightly less than 1 inch from the left edge of the paper. The **.po 0** request sets the offset as far to the left as it will go.

The indent request (**.in**) causes the left margin to be indented by some specified amount from the page offset. If the **.in** request is used to move the left margin to the right and the **.ll** request is used to move the right margin to the left, offset blocks of text are obtained. As an example:

```
.in 0.5i
.ll -0.5i
text to be set into a block
.ll +0.5i
.in -0.5i
```

will create a block of text that looks like:

> A clergyman at Cambridge preached a sermon which one of his auditors commended. "Yes," said a gentleman to whom it was mentioned, "it was a good sermon, but he stole it." This was told to the preacher. He resented it, and called the gentleman to retract what he had said. "I am not," replied the aggressor, "very apt to retract my words, but in this instance I will. I said, you had stolen the sermon; I find I was wrong; for on returning home and referring

> to the book whence I thought it was taken, I found it there."

The use of + and − changes the previous setting by the specified amount rather than just overriding it. The distinction is quite important:

- **.ll +1i** makes lines 1 inch longer
- **.ll 1i** makes lines 1 inch long.

With the **.in**, **.ll**, and **.po** requests, the previous value is used if no argument is specified.

The **.ti** request is used to temporarily indent a single line. The default unit for **.ti**, as for most horizontally oriented requests (**.ll**, **.in**, **.po**), is ems. An em is roughly the width of the letter **m** in the current point size. Precisely, an em in size *p* is *p* points. Although inches are usually clearer than ems to people who do not set type for a living, ems have a place: they are a measure of size that is proportional to the current point size. The ems unit is used to make text that keeps its proportions regardless of point size. The ems can be specified as scale factors directly, as in **.ti 2.5m**.

Lines can be indented negatively if the indent is already positive:

```
.ti -.3i
```

causes the next line to be moved back 3/10 of an inch.

To make a decorative initial capital that is three lines high:

- The whole paragraph is indented.
- The initial character is moved back with the **.ti** request.
- The initial character is made bigger (e.g., **\s36N\s0**) and moved down from its normal position.

4. Tabs

Tabs (the ASCII **horizontal tab** character) can be used to produce output in columns or to set the horizontal position of output. Typically, tabs are used only in unfilled text. Tab stops are set by default every half inch from the current indent but can be changed by the **.ta** request. Tab stops are set every inch, for example, with the following entry:

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

Tab stops are left justified (as on a typewriter), so lining up columns of right-justified numbers can be a problem. If there are many numbers or if a table layout is needed, the table formatting program (**tbl**) is available.

A handful of numeric columns can be produced by preceding every number with enough blanks to make it line up when typed. For instance:

```
.nf
.ta 1i 2i 3i
\0\01Ⓣ\0\02Ⓣ\0\03
\040Ⓣ\050Ⓣ\060
700Ⓣ 800Ⓣ 900
.fi
```

Each leading blank is a **\0** escape sequence. This character does not print but has the same width as a digit. The Ⓣ symbol represents a tab character. When printed, the above input produces:

```
  1          2          3
 40         50         60
700        800        900
```

It is also possible to fill up tabbed-over space with some character other than blanks by setting the tab replacement character with the **.tc** request:

```
.ta 2i 3i
.tc \(ru \" the "\(ru" string is the rule ( _ ) character
NameⓉAgeⓉ
```

produces:

```
Name _________________Age ________
```

To reset the tab replacement character to a blank, the **.tc** request (with no argument) is used. Lines can also be drawn with the **\l** escape sequence as described in paragraph 5.4.

The **troff** processor provides a general mechanism called "fields" for setting up complicated columns. This is used by the **tbl** program.

5. Local Motions

The **troff** processor provides a number of escape sequences for placing characters of any size at any place. They can be used to draw special characters or to tune the output for a particular appearance. Most of these sequences are straightforward but messy to read and tough to type correctly.

5.1 Vertical Motions

If the **eqn** program is not used, subscripts and superscripts are most easily done with the half-line local motions **\u** and **\d** sequences. To go back up the page half a point size, insert a **\u** at the desired place; to go down half a point size, insert a **\d**. The **\u** and **\d** should always be used in pairs. Since **\u** and **\d** refer to the current point size, they should either be both inside or both outside the size changes. Otherwise, an unbalanced vertical motion will result.

Sometimes the space given by **\u** and **\d** is not the right amount. The **\v** sequence can be used to request an arbitrary amount of vertical motion. The in-line sequence **\v'***N***'** causes motion up or down the page by the amount specified in *N*. For example, to move the character "N" down, the following would apply

```
.in +0.6i       \" indent paragraph
.ll -0.3i       \" shorten lines
.ti -0.3i       \" move N back
\v'2'\s36N\s0\v'-2'ott  met Shott, Nott
shot at Shott...
```

A minus sign causes upward motion, while no sign or a plus sign means down the page. Thus **\v'-2'** causes an upward vertical motion of two line spaces.

There are other ways to specify the amount of motion:

```
\v'0.1i'
\v'3p'
\v'-0.5m'
```

are all legal. The scale specifier **i**, **p**, or **m** goes inside the quotes. Any character can be used in place of the quotes. This is true of all other **troff** formatter commands and sequences described in this section.

Since the **troff** formatter does not take within-the-line vertical motions into account when figuring where it is on the page, output lines can have unexpected positions if the left and right ends are not at the same vertical position. Thus **\v**, like **\u** and **\d**, should always balance upward vertical motion in a line with the same amount in the downward direction.

5.2 Horizontal Motions

Arbitrary horizontal motions are also available, **\h** is analogous to **\v**, except that the default scale factor is ems instead of line spaces. As an example,

```
\h'-0.1i'
```

causes a backward motion of a tenth of an inch. In a practical situation, when printing the mathematical symbol > >, the default spacing is too wide, so **eqn** replaces this by

```
>\h'-0.3m'>
```

to produce >>.

Frequently, **h** is used with the "width function" **w** to generate motions equal to the width of some character string. The construction

```
\w'thing'
```

is a number equal to the width of "thing" in machine units (1/432 inch). All **troff** formatter computations are ultimately done in these units. To move horizontally, the width of an **x**,

```
\h' \w'x'u'
```

is used. Since the default scale factor for all horizontal dimensions is **m** (ems), **u** (machine units) must be used, or the motion produced will be too large. Nested quotes are acceptable to the **troff** formatter as long as none are omitted. An example of this kind of construction would be to print the string **.sp** by overstriking with a slight offset. The following example prints **.sp**, moves left by the width of **.sp**, moves right one unit, and prints **.sp** again:

```
.sp\h'-\w'.sp'u'\h'1u'.sp
```

There are several special-purpose **troff** formatter sequences for local motion:

- The **0** is an unpaddable (never widened or split across a line by line justification and filling) white space the same width as a digit.
- The \<**space**> is an unpaddable character the width of a space.
- The \| is 1/6 em wide.
- The \^ is 1/12 em wide.
- The **&** has zero width and is useful in entering a text line that would otherwise begin with a **.**.

- The **\o** sequence causes up to nine characters to be overstruck, centered on the widest. This is for accents such as:

```
syst\o"e\(ga"me t\o"e\(aa"l\o"e\(aa"phonique
```

which produces

```
systéme téléphonique
```

The accents **\(ga** and **\(aa** (\´ and \´) are just one character to the **troff** formatter.

5.3 Overstrikes

Overstrikes can be made with another special convention, **\z**, the zero-motion sequence. Normal horizontal motion is suppressed with the **\zx** after printing the single character **x**, so another character can be laid on top of it. Although sizes can be changed within **\o**, characters are centered on the widest, and there can be no horizontal or vertical motions. The **\z** may be the only way to get what is needed.

A more ornate overstrike is given by the bracketing function **\b**, which piles up characters vertically, centered on the current baseline. Thus big brackets are obtained by constructing them with piled-up smaller pieces.

5.4 Drawing Lines

A convenient facility for drawing horizontal and vertical lines of arbitrary length with arbitrary characters is provided by the **troff** and **otroff** formatters. A 1-inch long line is printed with a **\l'1i'** sequence. The length can be followed by the character to use if the _ is not appropriate. The **\l'0.5i.'** sequence draws a 1/2 inch line of dots. Escape sequence **\L** is analogous, except that it draws a vertical instead of a horizontal line.

The **troff** formatter provides an even better facility for drawing lines using the \D escape sequence. This funtion can also be used to draw arcs, circles, and ellipses.

6. Strings

If a paper contains a large number of occurrences of an acute accent over a letter **e**, typing **\o" e\'"** for each **é** would be a nuisance. Fortunately, the **troff** formatter provides a way to store an arbitrary collection of text in a "string", and thereafter use the string name as a shorthand for its contents. Strings are one of several **troff** formatter mechanisms whose judicious use permits typing a document with less effort and organizing it so that extensive format changes can be made with few editing changes.

A reference to a string is replaced by whatever text the string was defined as. Strings are defined with the **.ds** request. The line

```
.ds e \o"e\' "
```

defines the string **e** to have the value \o" e\'" .

String names may be either 1- or 2-characters long. They are referred to by ***x** for 1-character names or ***(xy** for 2-character names. Thus to get

```
téléphone
```

given the definition of the string **e** as above,

```
t\*el\*ephone
```

is the input.

If a string must begin with blanks, it is defined as

```
.ds xx "       text
```

The double quote signals the beginning of the definition. There is no trailing quote; the end of the line terminates the string.

A string may be several lines long. If the **troff** formatter encounters a \ at the end of any line, it is thrown away and the next line is added to the current one. A long string can be made by ending each line except the last with a backslash:

```
.ds xx this \
is a very \
long string
```

Strings may be defined in terms of other strings or even in terms of themselves.

7. Introduction to Macros

In its simplest form, a macro is a shorthand notation similiar to a string. For instance, if every paragraph is to start in exactly the same way, with a space and a temporary indent of two ems, the following requests would perform the operation:

```
.sp
.ti +2m
```

To save typing these requests every time used, they could be collapsed into one shorthand line, such as a **troff** command, **.PP**. The **.PP** is called a *macro*. The way to tell the **troff** formatter what **.PP** means is to define it with the **.de** request:

```
.de PP
.sp
.ti +2m
..
```

The first line names the macro (**.PP** in this example). It is in uppercase so it will not conflict with any name that the **troff** formatter might already know about. The last line (..) marks the end of the definition. In between is the text which is inserted whenever the **troff** formatter encounters the **.PP** macro call. A macro can contain any mixture of text and formatting requests.

The definition of a macro has to precede its first use; undefined macros are ignored. Names are restricted to one or two characters.

Using macros for commonly occurring sequences of requests is important since it saves typing and makes later changes easier. If it is decided that in producing a document the paragraph indent is too small, the vertical space is too large, and Roman font should be forced, only the definition of **.PP** needs to be changed to read

```
.de PP          \" paragraph macro
.sp 2p
.ti +3m
.ft R
..
```

The change takes effect everywhere **.PP** is used and is easier than changing commands throughout the whole document.

A **troff** formatter escape sequence that causes the rest of the line to be ignored is \" . It is used to add comments to the macro definition (a wise idea once definitions get complicated).

Another example of macros is this pair that start and end a block of offset, unfilled text

```
.de OS          \" start indented block
.sp
.nf
.in +0.5i
..
.de OE          \" end indented block
.sp
.fi
.in -0.5i
..
```

The **.OS** and **.OE** macros could be used before and after text to provide the following effect:

```
Copy to
John Doe
Richard Roberts
Stanley Smith
```

In this example, the indention used is **.in +0.5i** instead of **.in 0.5i**. This permits the nesting of the **.OS** and **.OE** macros to get blocks within blocks.

Should the amount of indention be changed at a later date, it is necessary to change only the definitions of **.OS** and **.OE**, not individual requests throughout the whole paper.

8. Titles, Pages, and Page Numbering

Titles, pages, and page numbering is a complicated area where nothing is done automatically. Of necessity, some of this section is a cookbook to be copied literally until some experience is obtained.

To get a title at the top of each page, such as:

```
left top                center top                right top
```

it was possible on an older system (**roff/sroff**, see Chapters 5 and 6) to get headers and footers automatically on every page with the following:

```
.he 'left top'center top'right top'
.fo 'left bottom'center bottom'right bottom'
```

This does not work in the **troff** formatter. Instead specifications must be provided:

- What to do at and around the title line
- When to print the title
- What the actual title is.

The **.NP** macro (new page) is defined to process titles at the end of one page and the beginning of the next:

```
.de NP
'bp
'sp 0.5i
.tl 'left top'center top'right top'
'sp 0.3i
..
```

These requests are explained as follows:

- The **'bp** (begin page) request causes a skip to the top-of-page.
- The **'sp 0.5i** request will space down 1/2 inch.
- The **.tl** request prints the title.
- The **'sp 0.3i** request provides another 0.3 inch space.

The reason that the **'bp** and **'sp** requests are used instead of the **.bp** and **.sp** requests is that the **.sp** and **.bp** cause a break to take place. This means that all the input text collected but not yet printed is flushed out as soon as possible, and the next input line is guaranteed to start a new line of output. Had **.bp** been used in the **.NP** macro, a break in the middle of the current output line would occur when a new page is started. The effect would be to print the left-over part of that line at the top of the page, followed by the next input line on a new output line. This is not desired. Using "'" instead of "." for a request tells the **troff** formatter that no break is to take place. The output line currently being filled should not be forced out before the space or new page.

The list of requests that cause a break is short and natural:

.bp	begin page
.br	break

.ce	center
.fi	fill mode
.nf	no-fill mode
.sp	space
.in	indent
.ti	temporary indent

Other requests cause no break, regardless of whether a "." or a "'" is used. If a break is really needed, a **.br** request at the appropriate place will provide it.

To ask for **.NP** at the bottom of each page, a statement like "when the text is within an inch of the bottom of the page, start the processing for a new page" is used. This is done with the **.wh** request. For example:

```
.wh -1i NP
```

No "." character is used before **NP** since it is simply the name of a macro and not a macro call. The minus sign means "measure up from the bottom of the page", so **-1i** means 1 inch from the bottom. The **.wh** request appears in the input data outside the definition of the **.NP** macro. Typically, the input would be

```
.de NP
    --- body of macro
..
.wh -1i NP
```

As text is actually being output, the **troff** formatter keeps track of its vertical position on the page; and after a line is printed within 1 inch from the bottom, the **.NP** macro is activated.

- The **.wh** request sets a trap at the specified place.
- The trap is sprung when that point is passed.

The **.NP** macro causes a skip to the top of the next page (that is what the **'bp** was for) and prints the title with appropriate margins.

Something to beware of when changing fonts or point sizes is crossing a page boundary in an unexpected font or size.

- Titles come out in the size and font most recently specified instead of what was intended.
- The length of a title is independent of the current line length, so titles will come out at the default length of 6.5 inches unless changed. Changing title length is done with the **.lt** request.

There are several ways to fix the problems of point sizes and fonts in titles. The **.NP** macro can be changed to set the proper size and font for the title, and then restore the previous values, like this:

```
.de NP
'bp
'sp 0.5i
.ft R          \" set title font to Roman
.ps 10         \" set size to 10 point
.lt 6i         \" set length to 6 inches
.tl 'left top'center top'right top'
.ps            \" revert to previous size
.ft P          \" and to previous font
'sp 0.3i
..
```

This version of **.NP** does not work if the fields in the **.tl** request contain size or font changes. To cope with that contingency requires the **troff** formatter "environment" mechanism discussed in paragraph 12.

To get a footer at the bottom of a page, the **.NP** macro should be modified. One option is to have the **.NP** macro do some processing before the **'bp** request. Another option is to split the **.NP** macro into a footer macro (invoked at the bottom margin) and a header macro (invoked at the top of page).

Output page numbers are computed automatically as each page is produced (starting at 1), but no numbers are printed unless explicitly requested. To get page numbers printed, the % character should be included in the **.tl** request at the position where the number is to appear. For example:

```
.tl ''- % -''
```

centers the page number inside hyphens. The page number can be set at any time with either a **.bp n** request (which immediately starts a new page numbered **n**) or with **.pn n** (which sets the page number for the next page but does not cause a skip to the new page). The **.bp +n** sets the page number to **n** more than its current value. The **.bp** request without an argument means **.bp +1**.

9. Number Registers and Arithmetic

The **troff** processor has a facility for doing arithmetic and defining and using variables with numeric values, called *number registers*. Number registers, like strings and macros, can be useful in setting up a document so it is easy to change later. They also serve for any sort of arithmetic computation.

Like strings, number registers have 1- or 2-character names. They are set by the **.nr** request and are referenced anywhere by **\nx** (1-character name) or **\n(xy** (2-character name).

There are quite a few predefined number registers maintained by the **troff** formatter; among them:

- % for the current page number
- **nl** for the current vertical position on the page
- **dy**, **mo**, and **yr** for the current day, month, and year
- **.s** and **.f** for the current size and font (the font is the number of a font position).

Any of these can be used in computations like any other register, but some, like **.s** and **.f**, cannot be changed with **.nr**.

An example of the use of number registers is in an older macro package where most significant parameters are defined in terms of the values of a handful of number registers. These include the point size for text, the vertical spacing, and the line and title lengths. To set the point size and vertical spacing, a user may input

```
.nr PS 9
.nr VS 11
```

The paragraph macro, **.PP**, is roughly defined as follows:

```
.de PP
.ps \\n(PS    \" reset size
.vs \\n(VSp   \" spacing
.ft R         \" font
.sp 0.5v      \" half a line
.ti +3m
..
```

This sets the font to Roman and the point size and line spacing to whatever values are stored in the number registers **PS** and **VS**.

The reason for two backslashes is to indicate that a backslash is really meant. When the **troff** formatter originally reads the macro definition, it peels off one backslash to see what is coming next. Two backslashes in the definition are required to ensure that a backslash is left in the definition when the macro is used. If only one backslash is used, point size and vertical spacing will be frozen at the time the macro is defined, not when it is used.

Protection with an extra layer of backslashes is needed only for **\n**, *****, **\$**, and \ itself. Things like **\s**, **\f**, **\h**, **\v**, etc. do not need an extra backslash since they are converted by the **troff** formatter to an internal code immediately upon detection.

Arithmetic expressions can appear anywhere that a number is expected. As an example:

```
.nr PS \\n(PS-2
```

decrements register **PS** by 2. Expressions can use the arithmetic operators +, -, *, /, % (mod), the relational operators >, >=, <, <=, =, != (not equal), and parentheses.

So far, the arithmetic has been straightforward; more complicated things are tricky.

- Number registers hold only integers. In the **troff** formatter, arithmetic uses truncating integer division just like Fortran.

- In the absence of parentheses, evaluation is done left-to-right without any operator precedence, including relational operators. Thus:

```
7*-4+3/13
```

  becomes **-1**.

Number registers can occur anywhere in an expression and so can scale indicators like **p**, **i**, **m**, etc. (but no spaces). Although integer division causes truncation, each number and its scale indicator is converted to machine units (1/432 inch for the C/A/T) before any arithmetic is done, so **1i/2u** evaluates to **0.5i** correctly.

The scale indicator **u** often has to appear when least expected, in particular when arithmetic is being done in a context that implies horizontal or vertical dimensions. For example, **.ll 7/2i** is not 3 ½ inches. Instead, it is really 7 ems/2 inches. When translated into machine units, it becomes 0. This is because the default units for horizontal parameters (like **.ll**) are ems. Another incorrect try is **.ll 7i/2**. The 2 is 2 ems, so 7i/2 is small, although not 0. The correct way to specify 3 ½ inches is **.ll 7i/2u**. A safe rule is to attach a scale indicator to every number, even constants.

For arithmetic done within a **.nr** request, there is no implication of horizontal or vertical dimension, so the default units are "units", and 7i/2 and 7i/2u mean the same thing. Thus:

```
.nr ll 7i/2
.ll \\n(llu
```

accomplishes what is desired as long as the **u** on the **.ll** request is included.

10. Macros With Arguments

Two things are needed to be able to define macros that can change from one use to the next according to parameters supplied as arguments:

1. When the macro is defined, it must be indicated that some parts will be provided as arguments when the macro is called.

2. When the macro is called, the actual arguments to be plugged into the definition must be provided.

An example would be to define a macro (**.SM**) that will print its argument two points smaller than the surrounding text.

```
.de SM
\s-2\\$1\s+2
..
```

The macro call would appear:

```
.SM SMALL
```

The argument ("SMALL" in this example) would then appear two points smaller than the rest of the print.

Within a macro definition, the symbol \\$**n** refers to the **n**th argument with which the macro was called. Thus \\$**1** is the string to be placed in a smaller point size when **.SM** is called.

A slightly more complicated version is the following definition of **.SM** which permits optional second and third arguments that will be printed in the normal size:

```
.de SM
\\$3\s-2\\$1\s+2\\$2
..
```

Arguments not provided when the macro is called are treated as empty. The macro call

```
.SM ABLE ),
```

would appear (with "ABLE" in smaller type)

```
ABLE),
```

The macro call

```
.SM BAKER ). (
```

produces the following (with "BAKER" in smaller print):

```
(BAKER).
```

It is convenient to reverse the order of arguments because trailing punctuation is much more common than leading. The number of arguments that a macro was called with is available in number register .$.

The macro, **.BD**, is used to make "bold Roman" for **troff** formatter command names in text. It combines horizontal motions, width computations, and argument rearrangement:

```
.de BD
\&\\$3\f1\\$1\h'-\w'\\$1'u+2u'\\$1\fP\\$2
..
```

The **h** and **w** escape sequences need no extra backslash. The **&** is there in case the argument begins with a period. Two backslashes are needed with the **$n** commands to protect one of them when the

macro is being defined. A second example will make this clearer. A **.SH** macro can be defined to produce automatically numbered section headings with the title in smaller size bold print. The use is

```
.SH "Section title . . ."
```

If the argument to a macro is to contain blanks, it must be surrounded by double quotes.

The definition of the **.SH** macro is

```
.nr SH 0              \" initialize section number
.de SH
.sp 0.3i
.ft B
.nr SH \\n(SH+1       \" increment number
.ps \\n(PS-1          \" decrease PS number
\\n(SH. \\$1          \" title
.ps \\n(PS            \" restore PS
.sp 0.3i
.ft R
..
```

The section number is kept in number register **SH**, which is incremented each time just before use.

Note: A number register may have the same name as a macro without conflict but a string may not.

A **n(SH** and **n(PS** was used instead of a **n(SH** and **n(PS**. Had **n(SH** been used, it would have yielded the value of the register at the time the macro was defined, not at the time it was used. Similarly, by using **n(PS**, the point size at the time the macro was called is obtained.

An example that does not involve numbers is the **.NP** macro (defined earlier) which had the request

```
.tl 'left top'center top'right top'
```

The fields could be made into parameters by using instead

```
.tl '\\*(LT'\\*(CT'\\*(RT'
```

The title comes from three strings called LT, CT, and RT. If these are empty, the title will be a blank line. Normally, CT would be set with

```
.ds CT - % -
```

to give just the page number between hyphens. A user could supply private definitions for any of the strings.

11. Conditionals

Suppose it is desired that the **.SH** macro leave two extra inches of space just before Section 1, but nowhere else. The cleanest way to do that is to test inside the **.SH** macro whether the section number is 1, and add some space if it is. The **.if** command provides the conditional test that can be added just before the heading line is output:

```
.if \\n(SH=1 .sp 2i        \" first section only
```

The condition after the **.if** request can be any arithmetic or logical expression. If the condition is logically true or arithmetically greater than zero, the rest of the line is treated as if it were text (a request in this case). If the condition is false, zero, or negative, the rest of the line is skipped.

It is possible to do more than one request if a condition is true. For example, if several operations are to be done prior to Section 1, the **.S1** macro is defined and invoked when Section 1 is almost complete (as determined by an **.if**).

```
.de S1
   --- processing for section 1
..
.de SH
   ---
.if \\n(SH=1 .S1
   ---
..
```

An alternate way is to use the extended form of the **.if** request, e.g.:

```
.if \\n(SH=1 \{--- processing
for section 1 ---\}
```

The braces, "\{" and "\}", must occur in the positions shown or unexpected extra lines will be in the output. The **troff** processor also provides an "if-else" construction.

A condition can be negated by preceding it with **!**. The same effect as above is obtained (but less clearly) by using

```
.if !\\n(SH>1 .S1
```

There are a handful of other conditions that can be tested with **.if**. For example:

```
.if e .tl 'left top'center top'right top'      \" Even page title
.if o .tl 'left top'center top'right top'      \" Odd page title
```

gives facing pages different titles, depending on whether the page number is even or odd, when used inside an appropriate new page macro.

Two other conditions are **t** and **n**, which tells whether the formatter is **troff** or **nroff**:

```
.if t troff stuff . . .
.if n nroff stuff . . .
```

String comparisons may be made in a **.if** request.

```
.if 'string1'string2' stuff
```

executes the program **stuff** if *string1* is the same as *string2*. The character separating the strings can be anything reasonable that is not contained in either string. The strings themselves can reference strings with "*", arguments with "\$", etc.

12. Environments

There is a potential problem when going across a page boundary: parameters like *size* and *font for a page title* may be different from those in effect in the text when the page boundary occurs. A general way to deal with this and similar situations is provided by the **troff** formatter.

There are three environments. Each has independently selectable versions of many parameters associated with processing, including size, font, line and title lengths, fill/no-fill mode, tab stops, and partially collected lines. Thus the titling problem may be solved by processing the main text in one environment and titles in another with its own suitable parameters.

The **.ev n** request shifts to environment **n** (**n** must be 0, 1, or 2). The **.ev** request with no argument returns to the previous environment. Environment names are maintained in a stack, so calls for different environments may be nested and unwound consistently.

If the main text is processed in environment 0 where the **troff** formatter begins by default, the *new page* macro, **.NP**, can then be modified to process titles in environment 1, e.g.:

```
.de NP
.ev 1       \" shift to new environment
.lt 6i      \" set parameters here
```

```
.ft R
.ps 10
   --- any other processing
.ev         \" return to previous environment
..
```

It is also possible to initialize the parameters for an environment outside the **.NP** macro, but the version shown keeps all the processing in one place and is easier to understand and change.

13. Diversions

There are numerous occasions in page layout when it is necessary to store some text for a period of time without actually printing it. Footnotes are the most obvious example. Text of the footnote usually appears in the input well before the place on the page is reached where it is to be printed. The place where it is output normally depends upon the magnitude of the footnote. This implies that there must be a way to process the footnote, at least enough to decide its size without printing it.

A mechanism called a diversion is provided by the **troff** formatter for doing this processing. Any part of the output may be diverted into a macro instead of being printed; and at some convenient time, the macro may be put back into the input.

The **.di xy** request begins a diversion. All subsequent output is collected into the macro **xy** until the **.di** request with no arguments is encountered. This terminates the diversion. Processed text is available at any time thereafter by giving the **.xy** request. The vertical size of the last finished diversion is contained in the built-in number register **dn**. For instance, to implement a keep-release operation so that text between the macros **.KS** and **.KE** will not be split across a page boundary (as for a figure or table), the following applies:

- When a **.KS** is encountered, the output is diverted to determine its size.
- When a **.KE** is encountered and if the diverted text will fit on the current page, it is printed there. If the diverted text does not fit on the current page, it is printed at the top of the next page.

The definitions of the **.KS** and **.KE** macros are as follows:

```
.de KS                          \" start keep
.br                             \" start fresh line
.ev 1                           \" collect in new environment
.fi                             \" make it filled text
.di XX                          \" collect in XX
..

.de KE                          \" end keep
.br                             \" get last partial line
.di                             \" end diversion
.if \\n(dn>=\\n(.t .bp          \" bp if does not fit
.nf                             \" bring it back in no-fill
.XX                             \" text
.ev                             \" return to normal environment
..
```

The number register **nl** indicates the current position on the output page. Since output was being diverted, it remains at its value when the diversion started. The **dn** register contains the amount of text in the diversion. The distance to the next trap is in the built-in register **.t**. It is assumed that the next trap is at the bottom margin of the page. If the diversion is large enough to go past the trap, the **.if** is satisfied; and a **.bp** request is issued. In either case, the diverted output is brought back with **.XX**. It is essential to bring it back in no-fill mode so the **troff** formatter will do no further processing on it.

This is not the most general keep-release operation nor is it robust in the face of all conceivable inputs. It would require more space than available to display it in full generality. This manual is not intended to teach everything about diversions, but to sketch out enough so that existing macro packages can be read with some comprehension.

TUTORIAL EXAMPLES

Although the **nroff** and **troff** formatters have by design a syntax reminiscent of earlier text processors with the intent of easing their use, it is usually necessary to prepare at least a small set of macro definitions to describe most documents. Such common formatting needs as page margins and footnotes are deliberately not built into the **nroff** and **troff** formatters. 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.

Examples in the following text are intended to be useful and somewhat realistic but will not necessarily cover all relevant contingencies. Explicit numerical parameters are used to make the examples 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 data that depends on whether the **troff** or **nroff** formatter is being used.

1. Page Margins

Header and footer macros are 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. A simple header and footer macro definition is

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

This example provides 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 pseudopage transition. In fill mode, the output line that springs the footer trap was typically forced out because some part or whole word did not 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 (**'**). When the header/footer design contains material requiring independent text processing, the environment may be switched to avoid interaction with running text.

A more realistic example follows:

```
.de hd                  \" header
.if t .tl '\(rn'\(rn'   \" troff cut mark
.if \\n%>1 \{\
'sp |0.5i-1             \" tl base at 0.5 inch
.tl ''- % -' '          \" centered page number
.ps                     \" restore size
.ft                     \" restore font
.vs \}                  \" restore vs
'sp |1.0i               \" space to 1.0 inch
.ns                     \" turn on no-space mode
..
.de fo                  \" footer
.ps 10                  \" set footer/header size
.ft R                   \" set font
.vs 12p                 \" set base-line spacing
.if \\n%=1 \{\
'sp |\\n(.pu-0.5i-1     \" tl base 0.5 inch up
.tl ''- % -' ' \}       \" first page number
'bp
..
.wh 0 hd
.wh -1i fo
```

This example sets the size, font, and base-line spacing parameters for the footer material. Parameters are restored to their original values when the header is completed. The material in this case is a page number at the bottom of the first page and at the top of the remaining pages. If the **troff** formatter is used, a cut mark is drawn in the form of *root-en*'s at each margin. The **sp**'s refer to absolute positions to avoid dependence on the base-line spacing. Another reason for the **sp** 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 base-line spacing. The *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.

The above 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 to 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 0.5 inch up
```

2. Paragraphs and Headings

Housekeeping associated with starting a new paragraph should be collected in a paragraph macro that does the desired preparagraph spacing, forces the correct font, size, base-line 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            \" temporary indent
..
```

The first break in **pg** will force out any previous partial lines and must occur before the **.vs** request. The forcing of font, size, base-line spacing, and indent 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 the **troff** formatter; a larger space, at least as big as the output device vertical resolution, would be more suitable in the **nroff** formatter. 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           \" initial 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 the **.af** request.

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
```

The 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 the **lp** macro ends with **c** so that it will become a part of the first line of the text that follows.

3. 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      \" initial 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** request, will be made where the 2-column output is to begin.

4. Footnote Processing

The footnote mechanism 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
```

The following macro definitions cause footnotes to be 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 does not completely fit in the available space:

```
.de hd                  \" header
  ---
.nr x 0 1               \" initial 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 environment 1
.nf                     \" retain vertical size
.FN                     \" footnotes
.rm FN                  \" delete it
.if '\\n(.z'fy' .di     \" end overflow diversion
.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 first, include separator
.fi                          \" fill mode
..
.de ef                       \" end footnote
.br                          \" finish output
.nr z \\n(.v                 \" save spacing
.ev                          \" pop environment
.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            \" it did not 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 position
.wh -\\nbu  fx               \" fx at footer position
.ch fo -\\nbu                \" conceal fx with fo
```

- The header macro (**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**.

- If the register **dn** indicates a leftover footnote, the **fz** macro is invoked to reprocess it.

- The footnote start macro (**fn**) begins a diversion (append) in environment 1 and increments the footnote count register **x**; if the count is one, the footnote separator macro (**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 spacing size in register **z**.

- Register **y** is decremented by the size of the footnote which is available in register **dn**.

- On the first footnote, register **y** is further decremented by the difference in vertical base-line spacings of the two environments. This prevents late triggering of the footer trap from causing the last line of the combined footnotes to overflow.

- The footer trap is set to the lower 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 no-fill 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 or not **fy** is empty.

- Both **fo** and **fx** macros 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 terminates the overflow diversion (if necessary) and zeros **x** to disable **fx**. This is because the uncertainty correction, together with a not-too-late triggering of the footer, can result in footnote macros finishing before reaching the **fx** trap.

5. Last Page

After the last input file has ended, **nroff** and **troff** formatters invoke the end macro, if any, and eject the remainder of the page.

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

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 will deposit a null partial word and effect another last page.

Chapter 3

NROFF AND TROFF USER MANUAL

PAGE

1. Introduction 3-1
2. Usage 3-3
3. NROFF/TROFF Reference Manual 3-9
4. Nroff/Troff Escape Sequences 3-48
5. Predefined General Number Registers 3-51
6. Predefined Read-Only Number Registers 3-52
7. Font Control Requests 3-54
8. Character Size Control Requests 3-55
9. Page Control Requests 3-56
10. Text Filling, Adjusting, and Centering Requests 3-58
11. Vertical Spacing Requests 3-60
12. Line Length and Indenting Requests 3-62
13. Macro, String, Diversion, and Trap Requests 3-63
14. Number Registers Requests 3-66
15. Tab, Leader, and Field Requests 3-67
16. Input/Output and Translation Requests 3-68
17. Hyphenation Requests 3-70
18. Three-Part Title Requests 3-71
19. Output Line Numbering Requests 3-72
20. Conditional Acceptance Requests 3-73
21. Environment Switching Request 3-75
22. Insertions From Standard Input Requests 3-76
23. Input/Output File Switching Requests 3-77
24. Miscellaneous Requests 3-78
25. Output and Error Messages Request 3-80

Chapter 3

NROFF AND TROFF USER MANUAL

1. Introduction

This chapter is a user guide and reference manual to the UNIX system text formatters **nroff**, **troff**, and **otroff**.

The **nroff** text formatter formats text for typewriter-like terminals.

The **troff** (*device independent*) formatter formats text destined to be printed on a phototypesetter, but is intended to be converted by a postprocessor into codes that will drive a particular phototypesetter.

The **otroff** (*old* troff) formatter formats text for the Wang Laboratories C/A/T phototypesetter only.

The **nroff**, **troff**, and **otroff** text processors accept lines of text mixed with lines of format control information. They format the text into a printable, paginated document having a user-designed style. These formatters offer unusual freedom in document styling including:

- Arbitrary style headers and footers
- Arbitrary style footnotes
- Multiple automatic sequence numbering for paragraphs and sections
- Multiple column output
- Dynamic font and point-size control
- Arbitrary horizontal and vertical local motions at any point
- Overstriking, bracket construction, and line drawing functions.

Since the **nroff** and **troff** (or **otroff**) formatters are reasonably compatible, it is usually possible to prepare input acceptable to either. Conditional input is provided that enables the user to embed input expressly destined for either program (**nroff** or **troff/otroff**). The **nroff** formatter can prepare output directly for a variety of terminal types and is capable of utilizing the full resolution of each terminal.

The **otroff** text processor is a text-formatting program for driving the Wang Laboratories C/A/T phototypesetter on the UNIX operating system. It is capable of producing high quality text. The C/A/T phototypesetter normally runs with four fonts containing Roman, italic, and bold letters, a full Greek alphabet, a substantial number of special characters, and mathematical symbols. Characters can be printed in a range of sizes and placed anywhere on the page.

The **troff** text formatter is a program for driving virtually any phototypesetter since its output is an ASCII code describing the position, font, size, etc., of characters to be typeset on a page. This output must be converted by another program, called a postprocessor, into codes a particular phototypesetter will understand. Parameters such as fonts, character sizes, special characters, depend on the phototypesetter being driven. See Chapter 4, *DEVICE INDEPENDENT TROFF*, for a more complete description.

> ***Note:*** Throughout this chapter, a reference to **troff** also means **otroff** unless otherwise indicated. Where there are differences between the two, the differences are pointed out. Refer to Chapter 4 for a complete description of the device independent troff text formatter.

Full user control over fonts, sizes, and character positions, as well as the usual features of a formatter (right-margin justification, automatic hyphenation, page titling and numbering, etc.) are provided by the **troff** processor. It also provides macros, arithmetic variables and operations, and conditional testing for complicated formatting tasks.

Numbers enclosed in braces ({ }) refer to paragraph numbers within this section. For example, this is paragraph {1}.

2. Usage

The general form of invoking the **nroff** or **troff** formatter at the UNIX operating system command level is

nroff *options files*
or
otroff *options files*
or
troff *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 sign (-) is taken to be a file name corresponding to the standard input. Input is taken from the standard input if no file names are given. Options may appear in any order so long as they appear before the files.

nroff, otroff AND troff OPTIONS

Option ***Effect***

−**o***list* Prints 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*
- An initial -*N* means from the beginning to page *N*
- A final *N*- means from page *N* to the end.

−**n***N* Number the first generated page *N*.

−**s***N* Stop every *N* pages. The **nroff** formatter will halt after every *N* pages (default *N*=1) to allow paper loading or changing and will resume upon receipt of a new line. The **otroff** formatter will stop the phototypesetter every *N* pages, produce a trailer to allow changing cassettes, and resume after the phototypesetter is restarted. When using

troff, it is probably preferable to use the **-s** option on the postprocessor if one exists.

−**m***name* Prepend the macro file

/usr/lib/tmac/tmac.*name*

to the input files. Multiple −**m** macro package requests on a command line are accepted and are processed in sequence.

−**c***name* Prepend the compacted macro files

/usr/lib/macros/cmp.[**nt**].[**dt**].*name*
and
/usr/lib/macros/ucmp.[**nt**].*name*

to the input files. Multiple −**c** macro package requests on a command line are accepted. The compacted version of macro package *name* will be used if it exists. If not, the **nroff/otroff** formatter will try the equivalent −**m***name* option instead. This option should be used instead of −**m** because it makes the **nroff/otroff** formatters execute significantly faster.

> ***Note:*** This option only applies to the **nroff** and **otroff** formatters. Compacted macros are not supported with the **troff** formatter.

−**r***aN* Set register *a* (one character) to *N*.

−**i** Read standard input after the input files are exhausted.

−**q** Invoke the simultaneous input/output mode of the **.rd** request.

−**z** Suppress formatted output. Only message output will occur (from **.tm** requests and diagnostics).

−**k***name* Produce a compacted macro package from this invocation of the **nroff/otroff** formatter. This option has no effect

if no **.co** request is used in the **nroff/otroff** formatter input. Otherwise, the compacted output is produced in files *d.name* and *t.name*.

> ***Note:*** This option applies to **nroff** and **otroff** only. Compacted macros are not supported with the **troff** formatter.

nroff ONLY OPTIONS

Option	*Effect*
–T*name*	Specify the name of the output terminal type. Currently defined names are:

- **37** (default) for the TELETYPE® Model 37.
- **tn300** for the GE TermiNet 300 (or any terminal without half-line capabilities).
- **300** for the DASI 300.
- **300s** for the DASI 300s.
- **450** for the DASI 450.
- **X9700** for the Xerox 9700 laser printer.
- **X** for the EBCDIC TX train printer.
- **2631** for the Hewlett-Packard 2631 printer in regular mode.
- **2631-c** for the Hewlett-Packard 2631 printer in compressed mode.
- **2631-e** for the Hewlett-Packard 2631 printer in expanded mode.
- **382** for the DCT-382 terminal.

- **4000a** for the Trendata 4000a terminal.
- **832** for the Anderson Jacobson 832 terminal.
- **lp** for (generic) printers that can underline and tab.

−e Produce equally spaced words in adjusted lines using full terminal resolution.

−h Use output tabs during horizontal spacing to speed output and to reduce output byte count. Device tab settings are assumed to be every eight nominal character widths. The default settings of logical input tabs are also every eight nominal character widths.

−u*n* Set the emboldening factor (number of character overstrikes) in the **nroff** formatter for the third font position (bold) to be *n* (zero if *n* is missing).

troff/otroff ONLY OPTIONS

Option	*Effect*

−t Direct output to the standard output instead of the phototypesetter.

> ***Note:*** This option only applies to the **otroff** formatter.

−f Refrain from feeding paper and stopping phototypesetter at the end of the run (**otroff** only).

−w Wait until phototypesetter is available if busy.

> ***Note:*** This option only applies to the **otroff** formatter.

−b Report whether phototypesetter is busy or available. No text processing is done.

> ***Note:*** This option only applies to the **otroff** formatter.

−a Send a printable approximation of the results in the ASCII character set to the standard output. This approximates a display of the document.

−p*N* Print all characters in point size *N* while retaining all prescribed spacings and motions to reduce phototypesetter elapsed time.

> ***Note:*** This option only applies to the **otroff** formatter.

−T*name* Specifies the intended output device (phototypsetter). The default output device is defined locally.

−F*dir* Font information is to be accessed from the directory dir/*dev***name** where **name** is the default output device. The default font information directory is /usr/lib/font/dev **name**.

> ***Note:*** This option only applies to the **troff** formatter.

Each option is invoked as a separate argument. For example:

```
nroff -o4,8-10  -T300s  -mabc  file1 file2
```

requests formatting of pages 4, 8, 9, and 10 of a document contained in the files named *file1* and *file2*, specifies the output terminal as a DASI 300s, and invokes the macro package *abc*.

Various preprocessors and postprocessors are available for use with the **nroff** and **troff** formatters:

- The equation preprocessors are **neqn** and **eqn** (for **nroff** and **otroff/troff** formatters, respectively).
- The table-construction preprocessor is **tbl**.
- The constant-width font preprocessor for the **otroff** formatter is **ocw**.

> ***Note:*** The **ocw** preprocessor is not needed with the **troff** formatter.

- The picture drawing preprocessor for the **troff** formatter is **pic**.

  Note: The **pic** preprocessor cannot be used with the **otroff** formatter.

- A reverse-line postprocessor for multiple-column **nroff** formatter output on terminals without reverse-line ability is **col**. The TELETYPE® Model 37 escape sequences that the **nroff** formatter produces by default are expected by **col**.

- The TELETYPE® Model 37-simulator postprocessor for printing **nroff** formatter output on a Tektronix 4014 is **4014**.

- The phototypesetter-simulator postprocessor for the **troff** formatter that produces an approximation of phototypesetter output on a Tektronix 4014 is **tc**. The **otc** postprocessor performs a similar function for the **otroff** formatter. For example, in:

  ```
  tbl files ¦ eqn -Tcat ¦ otroff -t [options] ¦ otc
  ```

  or

  ```
  tbl files ¦ eqn ¦ troff [options] ¦ tc
  ```

  the first ¦ indicates the piping of **tbl** output to **eqn** input; the second ¦ indicates the piping of **eqn** output to the **otroff/troff** formatter input; and the third ¦ indicates the piping of the **otroff/troff** formatter output to the **otc/tc** postprocessor.

The **troff** formatter depends on a postprocessor to convert its output into codes for a particular phototypesetter. Currently, the only supported postprocessor for this purpose is a program called **daps**, for the Autologic APS-5 phototypesetter. There are two postprocessors that prepare **troff** output for printing on high quality laser printers. These are **dx9700** for the Xerox 9700 and **di10** for the Imagen Imprint-10. For more information about the preprocessors, refer to the *Preprocessors Reference*—Select Code 307-153.

3. NROFF/TROFF Reference Manual

3.1 General Explanation

3.1.1 Form of Input

Input data consists of *text lines* destined to be printed mixed with *control lines* that set parameters or otherwise control subsequent processing. Control lines begin with a control character, normally a period or an acute accent, followed by a 1- or 2-character name that specifies a basic request or the substitution of a user-defined macro in place of the control line. The acute accent control character suppresses the break function (the forced output of a partially filled line) caused by certain requests. Control characters may be separated from request/macro names by white space (spaces and/or tabs) for increased readability. Names must be followed by either a space or a newline character. Control lines with unrecognized request/macro names are ignored.

Various special functions may be introduced anywhere in the input by means of an escape character (\). For example, the function \n*R* causes the interpolation of the contents of the number register *R* in place of the function. Number register *R* is either *x* for a single letter register name or (*xx* for a 2-character register name. Part 4 , *Nroff/Troff Escape Sequences,* itemizes escape sequences for characters, indicators, and functions.

3.1.2 Formatter and Device Resolution

The **otroff** text processor internally uses 432 units/inch, corresponding to the C/A/T phototypesetter which has a horizontal resolution of 1/432 inch and a vertical resolution of 1/144 inch. The **troff** text processor uses the resolution of the phototypesetter for which its output is being prepared (723 units/inch for the APS-5). Both formatters rounds horizontal/vertical numerical parameter input to the actual horizontal/vertical resolution of the typesetter.

The **nroff** text processor internally uses 240 units/inch, corresponding to the least common multiple of the horizontal and vertical resolutions of various typewriter-like output devices. It rounds numerical input to the actual resolution of the output device indicated by the −**T** option (default TELETYPE® Model 37).

3.1.3 Numerical Parameter Input

Both **nroff** and **troff** formatters accept numerical input with the appended scale indicators shown in Figure 3-1, where *S* is the current type size in points, *V* is the current vertical line spacing in basic units, and *C* is a nominal character width in basic units.

SCALE INDICATOR	*MEANING*	*NUMBER OF BASIC UNITS*	
		troff	*nroff*
i	Inch	432*	240
c	Centimeter	432x50/127	240x50/127
P	Pica = 1/6 inch	72	240/6
m	em = *S* points	6x*S*	*C*
n	en = em/2	3x*S*	*C*, same as *em*
p	Point = 1/72 inch	6	240/72
u	Basic unit	1	1
v	Vertical line space	*V*	*V*
none	Default		

* 723 units/inch for the **troff** formatter driving the Autologic APS-5 phototypesetter. This value may vary for other phototypesetters.

Figure 3-1. Nroff/Troff Scale Indicators

In the **nroff** processor, both **em** and **en** are taken to be equal to *C*, which is output-device dependent; common values are 1/10 and 1/12 inch. Actual character widths in the **nroff** formatter need not be all the same. Constructed characters (such as ->) are often extra wide. Default scaling is

- **em** for horizontally oriented requests (**.ll**, **.in**, **.ti**, **.ta**, **.lt**, **.po**, **.mc**) and functions (**h**, **l**).
- **V** for vertically oriented requests (**.pl**, **.wh**, **.ch**, **.dt**, **.sp**, **.sv**, **.ne**, **.rt**) and functions (**v**, **x**, **L**).
- **p** for **.vs** request.
- **u** for **.nr**, **.if**, and **.ie** requests.

All other requests ignore scale indicators. When a number register containing an already appropriately scaled number is interpolated to provide numerical input, the basic 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.

The absolute position indicator (|) may be prepended 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 {3.7} 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.

3.1.4 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. 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 ½ the sum of 4.25 inches + 2 picas + 3 ems (30 points since the point size is 10).

3.1.5 Notation

Numerical parameters are indicated in this chapter in two ways. A $\pm N$ means that the argument may take the forms N, $+N$, or $-N$ and that the corresponding effect is to set the affected 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 **.ps**, **.ft**, **.po**, **.vs**, **.ls**, **.ll**, **.in**, and **.lt** requests restore the previous parameter value in the absence of an argument.

Single character arguments are indicated by single lowercase letters and 1- or 2-character arguments are indicated by a pair of lowercase letters. Character string arguments are indicated by multicharacter mnemonics.

3.2 Font and Character Size Control

3.2.1 Fonts

Default mounted fonts with the **otroff** formatter are Times Roman (**R**), Times Italic (**I**), Times Bold (**B**), and Special Mathematical (**S**) on physical typesetter positions 1, 2, 3, and 4, respectively. These font styles are shown in Figure 3-2. In Figure 3-2, the font examples are printed in 12-point, with a vertical spacing of 14-point, and with non-alphanumeric characters separated by ¼ em space. The original Special Mathematical Font was prepared for Bell Laboratories by Wang Laboratories, Inc., of Hudson, New Hampshire. The Times Roman, Italic, and Bold are among the many standard fonts available.

Times Roman

abcdefghijklmnopqrstuvwxyz
ABCDEFGHIJKLMNOPQRSTUVWXYZ
1234567890
! $ % & ( ) ‘ ’ * + – . , / : ; = ? [ ] ¦
● □ — - _ ¼ ½ ¾ fi fl ff ffi ffl ° † ′ ¢ ® ©

Times Italic

abcdefghijklmnopqrstuvwxyz
ABCDEFGHIJKLMNOPQRSTUVWXYZ
1234567890
*! $ % & ( ) ‘ ’ * + – . , / : ; = ? [ ] ¦*
● □ — - _ ¼ ½ ¾ fi fl ff ffi ffl ° † ′ ¢ ® ©

Times Bold

abcdefghijklmnopqrstuvwxyz
ABCDEFGHIJKLMNOPQRSTUVWXYZ
1234567890
! $ % & ( ) ‘ ’ * + – . , / : ; = ? [ ] ¦
● □ — - _ ¼ ½ ¾ fi fl ff ffi ffl ° † ′ ¢ ® ©

Special Mathematical Font

" ´ \ ^ _ § ~ / < > { } # @ + – =*
α β γ δ ε ζ η θ ι κ λ μ ν ξ ο π ρ σ s τ υ φ χ ψ ω
Γ Δ Θ Λ Ξ Π Σ Υ Φ Ψ Ω
√ ¯ ≥ ≤ ≡ ∼ ≃ ≠ → ← ↑ ↓ × ÷ ± ∪ ∩ ⊂ ⊃ ⊆
⊇ ∞ ∂
§ ∇ ∫ ∝ ∅ ∈ ‡ ☞ ☜ ¦ ◯ ⌠ ⌡ ⌋ ⌊ ⌉ ⌈ ⌊ ⌋ ⌈ ⌉

Figure 3-2. Some Font Style Examples

The default fonts available with the **troff** formatter depend on the intended phototypesetter. Refer to Chapter 4, *DEVICE INDEPENDENT TROFF*, for font style examples for the Autologic APS-5 phototypesetter.

The current font may be changed by use of the **.ft** request or by embedding at any desired point either **f***x*, **f**(*xx*, or **f***N*, where *x* and *xx* are the name of a font and *N* is a numerical font position. For the **otroff** formatter, the font name must already be mounted in a font position. With the **troff** formatter, the named font is loaded on font position 0 if the font exists and is not currently mounted by default or by a **.fp** request, but the font must still or again be in position 0 when the line is printed.

It is not necessary to change to the Special Font; characters on that font are automatically handled. With the **otroff** formatter, a request for a named but not mounted font is ignored.

The **troff** text processor can be informed that any particular font is to be mounted by use of the **.fp** request. The list of known fonts is installation dependent.

Font control is understood by the **nroff** formatter which normally underlines italic characters. Part 7 of this chapter contains a summary and explanation of font control requests.

In the subsequent discussion of font-related requests, *F* represents either a 1- or 2-character font name or the numerical font position, 1 through 4. The current font is available as numerical position in the read-only number register **.f**.

3.2.2 Character Set

The **troff** character set consists of the so-called Commercial II character set plus a Special Mathematical font character set each having 102 characters. All ASCII characters are included with some on the Special Mathematical font. The ASCII characters are input as themselves (with three exceptions); and non-ASCII characters are input in the form \(*xx*, where *xx* is a 2-character name given in Figure 3-4 and Figure 3-5. The three ASCII character exceptions are mapped as shown in Figure 3-3.

ASCII INPUT		*PRINTED BY troff*	
CHARACTER	*NAME*	*CHARACTER*	*NAME*
´	acute accent	’	close quote
`	grave accent	‘	open quote
-	minus	-	hyphen

Figure 3-3. Troff ASCII Character Mapping

The characters ´, `, and − may be input by \', \`, and \−, respectively, or by their names. The ASCII characters @, #, ", ´, `, <, >, \, {, }, ~, ^, and _ exist on the Special Mathematical font and are printed as a one em space if that font is not mounted.

The **nroff** text processor understands the entire **troff** character set but can print only:

- ASCII characters.
- Additional characters as may be available on the output device.
- Such characters as may be able to be constructed by overstriking or other combinations.
- Those characters that can reasonably be mapped into other printable characters.

The exact behavior is determined by a driving table prepared for each device. The characters ´, `, and _ print as themselves.

CHARACTER	*INPUT NAME*	*CHARACTER NAME*
’	’	close quote
‘	`	open quote
—	\(em	¾ Em dash
-	-	hyphen or
-	\(hy	hyphen
−	\-	current font minus
•	\(bu	bullet
□	\(sq	square
_	\(ru	rule
¼	\(14	¼
½	\(12	½
¾	\(34	¾
fi	\(fi	fi
fl	\(fl	fl
ff	\(ff	ff
ffi	\(Fi	ffi
ffl	\(Fl	ffl
°	\(de	degree
†	\(dg	dagger
′	\(fm	foot mark
¢	\(ct	cent sign
®	\(rg	registered
©	\(co	copyright

Figure 3-4. Naming Conventions for Non-ASCII Characters on the Standard Fonts

CHARACTER	*INPUT NAME*	*CHARACTER NAME*
+	\(pl	math plus
-	\(mi	math minus
=	\(eq	math equals
*	\(**	math star
§	\(sc	section
´	\(aa	acute accent
`	\(ga	grave accent
_	\(ul	underrule
/	\(sl	slash (matching backslash)
α	\(*a	alpha
β	\(*b	beta
γ	\(*g	gamma
δ	\(*d	delta
ε	\(*e	epsilon
ζ	\(*z	zeta
η	\(*y	eta
θ	\(*h	theta
ι	\(*i	iota
$\varkappa$	\(*k	kappa
λ	\(*l	lambda
μ	\(*m	mu
ν	\(*n	nu
ξ	\(*c	xi
o	\(*o	omicron
π	\(*p	pi
ρ	\(*r	rho
σ	\(*s	sigma
ς	\(ts	terminal sigma
τ	\(*t	tau
υ	\(*u	upsilon
ϕ	\(*f	phi
χ	\(*x	chi
ψ	\(*q	psi
ω	\(*w	omega

Figure 3-5. Naming Conventions for Non-ASCII Characters on the Special Font (Sheet 1 of 3)

CHARACTER	*INPUT NAME*	*CHARACTER NAME*
A	\(*A	Alpha†
B	\(*B	Beta†
Γ	\(*G	Gamma
Δ	\(*D	Delta
E	\(*E	Epsilon†
Z	\(*Z	Zeta†
H	\(*Y	Eta†
Θ	\(*H	Theta
I	\(*I	Iota†
K	\(*K	Kappa†
Λ	\(*L	Lambda
M	\(*M	Mu†
N	\(*N	Nu†
Ξ	\(*C	Xi
O	\(*O	Omicron†
Π	\(*P	Pi
P	\(*R	Rho†
Σ	\(*S	Sigma
T	\(*T	Tau†
Υ	\(*U	Upsilon
Φ	\(*F	Phi
X	\(*X	Chi†
Ψ	\(*Q	Psi
Ω	\(*W	Omega
√	\(sr	square root
‾	\(rn	root en extender
≥	\(>=	>=
≤	\(<=	<=
≡	\(==	identically equal
≃	\(~=	approximately equal
~	\(ap	approximates
≠	\(!=	not equal
→	\(->	right arrow
←	\(<-	left arrow

† Mapped into uppercase English letters on the font mounted on font position one.

Figure 3-5. Naming Conventions for Non-ASCII Characters on the Special Font (Sheet 2 of 3)

CHARACTER	*INPUT NAME*	*CHARACTER NAME*
↑	\(ua	up arrow
↓	\(da	down arrow
×	\(mu	multiply
÷	\(di	divide
±	\(+-	plus-minus
∪	\(cu	cup (union)
∩	\(ca	cap (intersection)
⊂	\(sb	subset of
⊃	\(sp	superset of
⊆	\(ib	improper subset
⊇	\(ip	improper superset
∝	\(if	infinity
∂	\(pd	partial derivative
∇	\(gr	gradient
	\(no	not
∫	\(is	integral sign
∝	\(pt	proportional to
∅	\(es	empty set
∈	\(mo	member of
|	\(br	box vertical rule
‡	\(dd	double dagger
☞	\(rh	right hand
☜	\(lh	left hand
∥	\(or	or
○	\(ci	circle
⎧	\(lt	left top (big brace)
⎩	\(lb	left bottom (big brace)
⎫	\(rt	right top (big brace)
⎭	\(rb	right bottom (big brace)
⎨	\(lk	left center (big brace)
⎬	\(rk	right center (big brace)
┃	\(bv	bold vertical
⌊	\(lf	left floor (big bracket)
⌋	\(rf	right floor (big bracket)
⌈	\(lc	left ceiling (big bracket)
⌉	\(rc	right ceiling (big bracket)

Figure 3-5. Naming Conventions for Non-ASCII Characters on the Special Font (Sheet 3 of 3)

3.2.3 Character Size

For the **otroff** formatter, character point sizes available are 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. Character sizes with the **troff** formatter depend on the phototypesetter installation. 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**$\pm N$ ($1 \leq N \leq 9$) 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 for **otroff**. With **troff**, requested point size values that are between two valid sizes yield the closer of the two. The current size is available in the **.s** number register. The **nroff** formatter ignores type size control. Part 8, *Character Size Control Requests*, contains a summary and explanation of character size requests.

3.3 Page Control

Top and bottom margins are not automatically provided. They may be defined by two macros which set traps at vertical positions 0 (top) and $-N$ (N from the bottom) {3.7.5}. A pseudo-page transition onto the first page occurs either when the first break occurs or when the first nondiverted text processing occurs. Arrangements for a trap to occur at the top of the first page must be completed before this transition. A summary and explanation of page control requests is shown in Part 9, *Page Control Requests*. References to the current diversion mean that the mechanism being described works during both ordinary and diverted output (the former is considered as the top diversion level).

Usable page width on the Wang C/A/T phototypesetter is about 7.54 inches. This may differ on other phototypesetters. The left margin begins about 1/27 inch from the edge of the 8-inch wide, continuous roll paper {4.4}. Physical limitations on the **nroff** text processor output are output-device dependent.

3.4 Text Filling, Adjusting, and Centering

3.4.1 Filling and Adjusting

Normally, words are collected from input text lines and assembled into an output text line until some word does not fit. An attempt may be made to hyphenate the word in an effort to assemble a part of it into the output line. The spaces between the words on the output line are 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* backslash-space character (\ ). The adjusted word spacings are uniform in the **troff** formatter, and the minimum interword spacing can be controlled with the **.ss** request. In the **nroff** formatter, 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 can all be prevented or controlled. The text length on the last line output is available in the **.n** number register, and text base-line position on the page for this line is in the **nl** number register. The text base-line high-water mark (lowest place) on the current page is in the **.h** register.

An input text line ending with **.**, **?**, **:**, or **!** is taken to be the end of a sentence, and an additional space character is automatically provided during filling. Multiple interword 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** escape sequence may be embedded in 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 to not look like a control line by prefacing it with the nonprinting, zero-width filler character (**\&**). Another way is to specify output translation of some convenient character into the control character using the **.tr** request.

3.4.2 Interrupted Text

Copying of an input line in no-fill mode can be interrupted by terminating the partial line with a **\c** escape sequence. 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.

Part 10 contains a summary and explanation of filling, adjusting, and centering requests.

3.5 Vertical Spacing

3.5.1 Base-Line Spacing

Vertical spacing size *(V)* between base lines of successive output lines can be set using the **.vs** request with a resolution of 1/144 inch = 1/2 point in the **otroff** formatter and to the output device resolution in the **nroff** and **troff** formatters. Spacing size must be large enough to accommodate character sizes on affected output lines. For the common type sizes (9 through 12 points), usual typesetting practice is to set *V* to two points greater than the point size; **troff** default is 10-point type on a 12-point spacing. The current *V* is available in the **.v** register. Multiple-*V* line separation (e.g., double spacing) may be obtained with a **.ls** request.

3.5.2 Extra Line Space

If a word contains a vertically 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. In this and other functions having a pair of delimiters around their parameter, the delimiter choice is arbitrary except that it cannot look like the continuation of a number expression for *N*.

- If *N* is negative, the output line containing the word will be preceded by *N* extra vertical spaces.

- If *N* is positive, the output line containing the word will be followed by *N* extra vertical spaces.

- If successive requests for extra space apply to the same line, the maximum value is used.

The most recently utilized post-line extra line space is available in the **.a** register.

3.5.3 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 the **.sv** request.

Part 11 contains a summary and explanation of vertical spacing requests.

3.6 Line Length and Indenting

The maximum line length for fill mode may be set with a **.ll** request. The indent may be set with a **.in** request; an indent applicable to only the next output line may be set with the **.ti** request. The line length includes indent space but not page offset space. The line length minus the indent is the basis for centering with the **.ce** request. If a partially collected line exists, the effect of **.ll**, **.in**, or **.ti** is delayed 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 3-part titles produced by **.tl** is independently set by **.lt**. Part 12 contains a summary and explanation of line length and indenting requests.

3.7 Macros, Strings, Diversions, and Position Traps

3.7.1 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 1- or 2-characters long and may usurp previously defined request, macro, or string names. Any of these entities may be renamed with **.rn** or removed with **.rm**.

- Macros are created by **.de** and **.di** and appended by **.am** and **.da** (**.di** and **.da** cause normal output to be stored in a macro).
- Strings are created by **.ds** and appended by **.as**.

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. The strings **x** and **xx** are interpolated at any desired point with ***x** and ***(xx**, respectively. String references and macro invocations may be nested.

3.7.2 Copy Mode Input Interpretation

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

- Contents of number registers indicated by **n** are interpolated.
- Strings indicated by * are interpolated {3.7.1}.
- Arguments indicated by \$ are interpolated.
- Concealed newline characters indicated by \<**newline**> are eliminated.
- Comments indicated by \" are eliminated {3.10.7}.
- **t** and **a** are interpreted as ASCII horizontal tab and start of heading (SOH), respectively {3.9.1}.

- \\ is interpreted as "\".
- \. is interpreted as ".".

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

3.7.3 Arguments

When a macro is invoked by name, the remainder of the line may contain up to nine arguments. The argument separator is the space character, 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. If the desired arguments will not fit on a line, a concealed newline character may be used to continue on the next line.

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 ($1 \leq N \leq 9$). 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.
```

The \$ was concealed in the definition with a prepended backslash. The number of currently available arguments is in the .$ register.

- No arguments are available at the top (nonmacro) level in this implementation.
- No arguments are available from within a string because string referencing is implemented as an input-level pushdown.
- No arguments are available within a trap-invoked macro.

Arguments are copied in copy mode onto a stack where they are available for reference. The mechanism does not allow an argument to contain a direct reference to a long string (interpolated at copy time), and it is advisable to conceal string references (with an extra \) to delay interpolation until argument reference time.

3.7.4 Diversions

Processed output may be diverted into a macro for purposes such as footnote processing 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 no-fill 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** request with the transparent mechanism described in paragraph 3.10.6.

Diversions may be nested and certain parameters and registers are associated with the current diversion level (the top nondiversion level may be thought of as diversion level 0). These parameters and registers are:

- Diversion trap and associated macro

- No-space mode
- Internally saved marked place (see **.mk** and **.rt**)
- Current vertical place (**.d** register)
- Current high-water text base line (**.h** register)
- Current diversion name (**.z** register).

3.7.5 Traps

Three types of trap mechanisms are available:

- Page trap
- Diversion trap
- Input-line-count trap.

Macro-invocation traps may be planted using **.wh** requests at any page position including the top. This trap position may be changed using the **.ch** request. 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. If the first planted trap 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.

Macro-invocation traps, effective in the current diversion, may be planted using **.dt** requests. The **.t** register works in a diversion. If there is no subsequent trap, a large distance is returned.

Part 13 contains a summary and explanation of macros, strings, diversion, and position traps requests.

3.8 Number Registers

A variety of predefined number registers (Part 5) are available to the user. In addition, the user may define his own named registers. Register names are 1- or 2-characters long and do not conflict with request, macro, or string names. Except for certain predefined read-only number registers (Part 6), 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.

Number registers are created and modified using the **.nr** request, which specifies name, numerical value, and automatic increment size. Registers are also modified if accessed with an automatic incrementing sequence. If the registers *x* and *xx* both contain *N* and have the automatic increment size *M*, Figure 2-6 shows the values interpolated for the indicated access sequences.

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*

Figure 3-6. Nroff/Troff Number Register Interpolation

According to the format specified by the **.af** request, a number register is converted (when interpolated) to:

- Decimal (default)
- Decimal with leading zeros
- Lowercase Roman
- Uppercase Roman
- Lowercase sequential alphabetic
- Uppercase sequential alphabetic.

Part 14 contains a summary and explanation of number registers requests.

3.9 Tabs, Leaders, and Fields

3.9.1 Tabs and Leaders

The ASCII horizontal tab character and the ASCII SOH character (the leader) 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 specified with a **.ta** request. 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 justified, right justified, and centered. In Figure 3-7:

- *next-string* consists of the input characters following the tab (or leader) up to the next tab (or leader) or end of line.
- *D* is the distance from the current position on the input line (where a tab or leader was found) to the next tab stop.
- *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 justified within *D*
Centered	*D-W/2*	Centered on right end of *D*

Figure 3-7. Nroff/Troff Tab Types

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 they may be used as *next-string* terminators.

Tabs and leaders are not interpreted in copy mode. The **\t** and **\a** always generate a noninterpreted tab and leader, respectively, and are equivalent to actual tabs and leaders in copy mode.

3.9.2 Fields

A field is contained between a pair of field delimiter characters. It 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 "^", then

```
#^xxx^right#
```

specifies a right-justified string with the string *xxx* centered in the remaining space.

Part 15 contains a summary and explanation of tab, leader, and field requests.

3.10 Input/Output Conventions and Character Translations

3.10.1 Input Character Translations

The newline character 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 a **.tr** request. All others are ignored.

The escape character (\ ) introduces sequences that cause the following character to mean another character or to indicate some function. A complete list of such sequences is given in Part 4. The escape character:

- should not be confused with the ASCII control character ESC of the same name.
- can be input with the sequence \\.
- can be changed with **.ec**, and all that has been said about the default \ becomes true for the new escape character.

A **e** sequence can be used to print the current escape character. If necessary or convenient, the escape mechanism may be turned off with **.eo** and restored with **.ec**. A summary and explanation of input character translations requests are contained in Part 16.

3.10.2 Ligatures

Five ligatures are available in the **troff** character set: fi, fl, ff, ffi, and ffl. They may be input (even in the **nroff** formatter) by \(fi, \(fl, \(ff, \(Fi, and \(Fl, respectively. The ligature mode is normally on in the **troff** formatter and automatically invokes ligatures during input. A summary and explanation of ligature requests are included in Part 16.

3.10.3 Backspacing, Underlining, and Overstriking

Unless in copy mode, the ASCII backspace character is replaced by a backward horizontal motion having the width of the space character. Underlining is a form of line drawing and, as a generalized overstriking function, is described in paragraph 3.12.

The **nroff** text processor underlines characters automatically in the underline font, specifiable with the **.uf** request. The underline font is normally on font position 2. In addition to **.ft** request and **f***F* escape sequence, the underline font may be selected by **.ul** and **.cu** requests. Underlining is restricted to an output-device-dependent subset of reasonable characters. A summary and explanation of backspacing, underlining, and overstriking requests are included in Part 16.

3.10.4 Control Characters

Both the *break* control character (.) and the *no-break* control character (') may be changed, if desired. 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. A summary and explanation of the **.cc** and **.c2** control character requests are included in Part 16.

3.10.5 Output Translation

One character can be made a stand-in for another character using the **.tr** request. 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. Graphic translation occurs at the moment of output (including diversion). Included in Part 16 is a summary and explanation of the output translation request.

3.10.6 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.

3.10.7 Comments and Concealed Newline Characters

An uncomfortably long input line that must stay on one line (e.g., a string definition or no-filled text) can be split into many physical lines by ending all but the last one with the escape character (\). The sequence \<**newline**> is ignored except in a comment. Comments may be embedded at the end of any line by prefacing them with \". The newline character 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 **.\"** .

3.11 Local Horizontal/Vertical Motion and Width Function

3.11.1 Local Motion

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 above and certain other escape sequences providing local motion are summarized and explained in Figure 3-8 and Figure 3-9. As an example, E^2 is generated by the sequence **E\v'-.5'\s-4\&2\s0\v'.5'**.

FUNCTION	*EFFECT IN*	
	TROFF	*NROFF*
\v'***N***'	Move distance ***N***	
\u	1/2 em up	1/2 line up
\d	1/2 em down	1/2 line down
\r	1 em up	1 line up

Figure 3-8. Vertical Local Motions

FUNCTION	*EFFECT IN*	
	TROFF	*NROFF*
\h'***N***'	Move distance ***N***	
\(space)	Unpaddable space-size space	
\0	Digit-size space	
\\|	1/6 em space	ignored
\^	1/12 em space	ignored

Figure 3-9. Horizontal Local Motions

3.11.2 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'1.'u
```

could be used to temporarily indent leftward a distance equal to the size of the string "**1.**".

The width function also sets three number registers. The registers **st** and **sb** are set to the highest and lowest extent of *string* relative to the baseline respectively; then, for example, the total height of the string is **n(stu-\n(sbu**. In the **troff** formatter, the number register **ct** is set to a value between 0 and 3:

- **0** means that all characters in *string* are short lowercase 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**).
- **3** means that both tall characters and characters with descenders are present.

3.11.3 Mark Horizontal Place

The escape sequence **k***x* will cause the current horizontal position in the input line to be stored in register *x*. As an example, the construction

```
\kx word\h'|\nxu+2u' word
```

will embolden *word* by backing up to almost its beginning and overprinting it, resulting in **word**.

3.12 Overstrike, Zero-Width, Bracket, and Line Drawing Functions

3.12.1 Overstrike

Automatically centered overstriking of up to nine characters is provided by the overstrike function **o'***string***'**. Characters in *string* are overprinted with centers aligned; the total width is that of the widest character. The *string* should not contain local vertical motion. As examples, "\o'e\´'" produces **é**, and "\o'\(ci\(pl'" produces ⊕.

3.12.2 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\(ci\(pl" will produce ⊕, and "\(br\z\(rn\(ul\(br" will produce the smallest possible constructed box (□).

3.12.3 Large Brackets

The Special Mathematical Font 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 one em and the total pile is centered one-half em above the current base line (one-half line in the **nroff** formatter). For example:

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

produces ⎡**E**⎤.

3.12.4 Line Drawing

The **l'***Nc***'** function will draw a string of repeated *c*'s toward the right for a distance *N* (**l** is lowercase L).

- If *c* looks like a continuation of an expression for *N*, it may be insulated from *N* with a "\&".

- If *c* is not specified, the base-line 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. In the case of characters that are designed to be connected, such as base-line rule (_), underrule (\(**ul**), and root en (\(**ru**), the remainder space is covered by overlapping. If *N* is less than the width of *c*, a single *c* is centered on a distance *N*. 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

.us " underlined words"

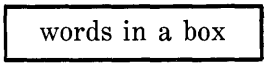
and

.bx " words in a box"

yield

underlined words

and



The function **L'Nc'** will draw a vertical line consisting of the optional character *c* stacked vertically apart one em (one line in **nroff**), with the first two characters overlapped, if necessary, to form a continuous line. The default character is box rule (**(br**); the other suitable character is bold vertical (**(bv**). The line is begun without any initial motion relative to the current base line. 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 base line 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 one-half em wide *underrule* were designed to form corners when using one em vertical spacings. For example, the macro

```
.de eb
.sp -1          \" compensate for next automatic base-line 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 *z* (e.g., using **.mk z**).

3.13 Hyphenation

The 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 prepended to suppress hyphenation. In addition, the user may specify a small exception word list. The default condition of hyphenation is off.

Only words that consist of a central alphabetic string surrounded by nonalphabetic strings (usually null) are considered candidates for automatic hyphenation. Words that were input containing hyphens (minus), em-dashes (**(em**), or hyphenation indicator characters

(such as mother-in-law) are always subject to splitting after those characters whether or not automatic hyphenation is on or off. Part 17 is a summary and explanation of hyphenation requests.

3.14 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**. The **.tl** may be used anywhere and is independent of the normal text collecting process. A common use is in header and footer macros. Part 18 is a summary and explanation of 3-part title requests.

3.15 Output Line Numbering

Automatic sequence numbering of output lines may be requested with **.nm**. When in effect, a 3-digit, Arabic number plus a digit space is prepended to output text lines. Text lines are 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 a **.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). Part 19 is a summary and explanation of output line numbering requests.

Figure 2-10 is an example of output line numbering. Paragraph portions are numbered with $M=3$.

- **.nm 1 3** was placed at the beginning.
- **.nm +0** was placed in front of the second and third paragraphs.
- **.nm** was 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 five greater than the last numbered line, with $M=5$, spacing S untouched, and the indent I set to 3.

Automatic sequence numbering of output lines may be requested with **.nm**. When in effect, a 3-digit, arabic number plus a digit-space is prepended to output text lines. Text lines are 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 a **.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). Part 19 is a summary and explanation of output line numbering requests.

As an example of output line numbering, paragraph portions of this figure 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** 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 five greater than the last numbered line, with *M*=5, spacing *S* untouched, and the indent *I* set to 3.

Figure 3-10. Example of Output Line Numbering

3.16 Conditional Acceptance of Input

Part 20 shows is a summary and explanation of conditional acceptance requests where:

- *c* is a 1-character, built-in condition name.
- ! signifies *not.*
- *N* is a numerical expression.
- *string1* and *string2* are strings delimited by any nonblank, non-numeric character not in the strings.
- *anything* represents what is conditionally accepted.

Built-in condition names are shown in Figure 3-11.

CONDITION NAME	*TRUE IF*
o	Current page number is odd
e	Current page number is even
t	Formatter is **troff**
n	Formatter is **nroff**

Figure 3-11. Nroff/Troff Built-In Condition Names

If condition *c* is true, if number *N* is greater than zero, or if 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 multiline case, the first line must begin with a left delimiter "\{ ", and the last line must end with a right delimiter "\} ". If the left delimiter is the last thing on that line, the following newline should be concealed with a "\" or a blank line may result on output.

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. The **.ie** - **.el** pairs may be nested. For example:

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

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
```

treats Page 1 differently from other pages.

3.17 Environment Switching

A number of parameters that control text processing are gathered together into an environment, that can be switched by the user. Environment parameters are those associated with some requests. Parts 7 through 25 of this section indicate in the "Explanation" column those requests so affected. 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. Part 21 is a summary and explanation of the environment switching request.

3.18 Insertions From Standard Input

The input can be switched temporarily to the system standard input with **.rd** and switched back when two newline characters in a row are found (the extra blank line is not used). This mechanism is intended for insertions in form-letter-like documentation. On the UNIX system, the standard input can be the user keyboard, a pipe, or a file.

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 insertions for all copies in one file to be used as the standard input and causing the file containing the letter to reinvoke itself by using the **.nx** request. The process would be ended by a **.ex** request in the insertion file. Part 22 is a summary and explanation of insertions from the standard input requests.

3.19 Input/Output File Switching

Part 23 is a summary and explanation of input/output file switching requests.

3.20 Miscellaneous

Part 24 is a summary and explanation of miscellaneous requests.

3.21 Output and Error Messages

Output from **.tm**, **.pm**, and prompt from **.rd**, as well as various error messages are written onto the UNIX system standard output and error (message) output. 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 the **nroff** and **troff** formatters. 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).
- *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 an * (in **nroff**) or a ☞ (in **troff**). The philosophy is to continue processing, if possible, on the grounds that output useful for debugging may be produced. If a serious error occurs, processing

terminates, and an appropriate message is printed. Examples are the inability to create, read, or write files, and the exceeding of certain internal limits that make future output unlikely to be useful. Part 25 is a summary and explanation of output and error messages requests.

3.22 Compacted Macros

> ***Note:*** The rest of this chapter applies only to the **nroff** and **otroff** formatters. Compacted macros are not supported by the **troff** formatter.

The time required to read a macro package by the **nroff** formatter may be lessened by using a compacted macro (a preprocessed version of a macro package). The compacted version is equivalent to the noncompacted version, except that a compacted macro package cannot be read by the **.so** request. A compacted version of a macro package, called *name*, is used by the **−c***name* command line option, while the uncompacted version is used by the **−m***name* option. Because **−c***name* defaults to **−m***name* if the *name* macro package has not been compacted, the user should always use **−c** rather than **−m**.

3.22.1 Building a Compacted Macro Package

Only macro, string, and diversion definitions; number register definitions and values; environment settings; and trap settings can be compacted. End macro (**em**) requests and any commands that may interact during package interpretation with command-line settings (such as references in the MM package to the number register **P**, which can be set from the command line) are not compactible. There are two steps to make a compacted macro from a macro package:

- Separate compactible from noncompactible parts
- Place noncompactible material at the end of the macro package with a **.co** request. The **.co** request indicates to the **nroff** formatter when to compact its current internal state.

Compactible Material

.
.
.

.co
Noncompactible Material

.
.
.

3.22.2 Produce Compacted Files

When compactible and noncompactible segments have been established, the **nroff** formatter may be run with the **−k** option to build the compacted files. For example, if the output file to be produced is called *mac*, the following may be used to build the compacted files:

nroff -k*mac mac*

This command causes the **nroff** formatter to create two files in the current directory, *d.mac* and *t.mac*.

> ***Note:*** When **nroff/otroff** is complied with the INCORE option (which is the default, except on the PDP*-11) only one file, *d.mac*, will be created. In this case, only *d.mac* should be installed, ignoring the missing *t.mac*.

The macro file must contain a **.co** request. Only lines before the **.co** request will be compacted. Both **−k** and **.co** are necessary. If no **.co** is found in the file, the **−k** is ignored. Likewise, if no **−k** appears on the command line, the **.co** is ignored.

Each macro package must be compacted separately by the **nroff** formatter. Compacted macro packages depend on the particular version of the **nroff** formatter that produced them. Any compacted macro packages must be recompacted when a new version of an **nroff**

---

* Trademark of Digital Equipment Corporation.

formatter is installed. If it is discovered that a macro package was produced by a different version than that attempting to read it, the **−c** will be abandoned and the equivalent **−m** option attempted instead.

3.22.3 Install Compacted Files

The two compacted files, *d.mac* and *t.mac*, must be installed into the system macro library **(/usr/lib/macros)** with the proper names. If the files were produced by an **nroff** formatter, **cmp.n.** must be prepended to their names. For example, if the macro package is called **mac**, the two **nroff** formatter compacted files may be installed by

```
cp d.mac /usr/lib/macros/cmp.n.d.mac
and
cp t.mac /usr/lib/macros/cmp.n.t.mac
```

3.22.4 Install Noncompactible Segment

The noncompactible segment from the original macro package must be installed on the system as

```
/usr/lib/macros/ucmp.[nt].mac
```

where **n** of [**nt**] means the **nroff** formatter version, and **t** means the **troff** formatter version. The noncompactible segment must be produced manually by using the editor. Using the **mac** package as an example, the following could be used to install the **nroff** formatter noncompactible segment:

```
$ ed mac
/^\.co$/+,$w /usr/lib/macros/ucmp.n.mac
```

4. Nroff/Troff Escape Sequences

\\	\ (to prevent or delay the interpretation of \ )
\´	Acute accent; equivalent to \(**aa**
\`	Grave accent; equivalent to \(**ga**
\-	Minus sign in the current font
\.	Period (dot) (see **de**)
\<space>	Unpaddable space-size space character
\0	Unpaddable digit width space
\\|	1/6 em narrow space character (zero width in the **nroff** formatter)
\^	1/12 em half-narrow space character (zero width in the **nroff** formatter)
\&	Nonprinting zero width character
\!	Transparent line indicator
\"	Beginning of comment
\$*N*	Interpolate argument ($1 \leq N \leq 9$)
\%	Default optional hyphenation character
\(*xx*	Character named *xx*
*x, *(*xx*	Interpolate string *x* or *xx*
\{	Begin conditional input
\}	End conditional input
\<newline>	Concealed (ignored) newline character

\a	Noninterpreted leader character
\b'*abc...*'	Bracket building function
\c	Continuation of interrupted text
\d	Forward (down) ½ em vertical motion (½ line in the **nroff** formatter)
\D'l dh dv'	Draw a line from the current position by *dh, dv.* (Not supported in **otroff**)
\D'c d'	Draw a circle of diameter *d* with left side at the current position. (Not supported in **otroff**)
\D'e d1 d2'	Draw an ellipse of diameters *d1* and *d2* with left side at current position. (Not supported in **otroff**)
\D'a dh1 dv1 dh2 dv2'	Draw a counterclockwise arc from current position to *dh1+dh2, dv1+dv2,* with center at *dh1, dv1* from current position. (Not supported in **otroff**)
\D'~ dh1 dv1 dh2 dv2 . . . '	Draw a B-spline from current position by *dh1, dv1,* then by *dh2, dv2,* then . . . . (Not supported in **otroff**)
\e	Printable version of current escape character
\f*x*, \f(*xx*, \f*N*	Change to font named *x* or *xx* or position *N*
\g*x*, \g(*xx*	Return the **.af-type** format of the register *x* or *xx* (returns nothing if *x* or *xx* has not yet been referenced)
\h'*N*'	Local horizontal motion; move right *N* (negative left)
\H'n'	Character heights are set to **n** points without changing widths. A height of the form ±n is an increment to the current point size; a height of 0 restores the height to the current point size. (Not supported in **otroff**)

\j*x*, \j(*xx*	Mark the current horizontal output position in register *x* or *xx*. This only exists with **otroff**.
\k*x*	Mark horizontal input place in register *x*
\l'*Nc*'	Horizontal line drawing function (optionally with *c*)
\L'*Nc*'	Vertical line drawing function (optionally with *c*)
\n*x*, \n(*xx*	Interpolate number register *x* or *xx*
\o'*abc...*'	Overstrike characters *a*, *b*, *c*...
\p	Break and spread output line
\r	Reverse 1 em vertical motion (reverse line in the **nroff** formatter)
\s*N*,\s±*N*	Point-size change function
\S'n'	Output is slanted *n* degrees. The value of n may be negative. A value of 0 turns slant mode off. (Not supported in **otroff**)
\t	Noninterpreted horizontal tab
\u	Reverse (up) ½ em vertical motion (½ line in the **nroff** formatter)
\v'*N*'	Local vertical motion; move down *N* (negative up)
\w'*string*'	Interpolate width of *string*
\x'*N*'	Extra line-space function (negative before, positive after)
\z*c*	Print *c* with zero width (without spacing)
\X	Any character not listed above

5. Predefined General Number Registers

%	Current page number.
ct	Character type (set by width function).
dl	Width (maximum) of last completed diversion.
dn	Height (vertical size) of last completed diversion.
dw	Current day of the week (1 through 7).
dy	Current day of the month (1 through 31).
hp	Current horizontal place on input line. (**otroff** only)
ln	Output line number.
mo	Current month (1 through 12).
nl	Vertical position of last printed text base line.
sb	Depth of string below base line (generated by width function).
st	Height of string above base line (generated by width function).
yr	Last two digits of current year.
c.	Provides general register access to the input line number in the current input file. Contains the same value as the read-only **.c** register.

6. Predefined Read-Only Number Registers

.$	Number of arguments available at the current macro level.
$$	Process-id of the troff process (**troff** only).
.A	Set to **1** in the **troff** formatter if **-a** option used; always **1** in the **nroff** formatter.
.F	Value is a *string* that is the name of the current input file.
.H	Available horizontal resolution in basic units.
.L	Contains the current line spacing parameter (the value of the most recent **.ls** request).
.P	Contains the value 1 if the current page is being printed and is zero otherwise, i.e., if the current page did not appear in the **−o** option list.
.T	Set to **1** in the **nroff** formatter if **-T** option used; always **0** in the **troff** formatter.
.V	Available vertical resolution in basic units.
.a	Post-line extra line space most recently utilized using **x**′*N*.
.b	Emboldening factor of the current font.
.c	Number of lines read from current input file.
.d	Current vertical place in current diversion; equal to **nl** if no diversion.
.f	Current font as physical quadrant (1 through 4).
.h	Text base-line high-water mark on current page or diversion.

.i	Current indent.
.j	Indicates the current adjustment mode and type. Can be saved and later given to the **.ad** request to restore a previous mode.
.k	Contains the horizontal size of the text portion (without indent) of the current partially collected output line, if any, in the current environment.
.l	Current line length.
.n	Length of text portion on previous output line.
.o	Current page offset.
.p	Current page length.
.R	Number of number registers that remain available for use.
.s	Current point size.
.t	Distance to the next trap.
.u	Equal to **1** in fill mode and **0** in no-fill mode.
.v	Current vertical line spacing.
.w	Width of previous character.
.x	Reserved version-dependent register.
.y	Reserved version-dependent register.
.z	Name of current diversion.

7. Font Control Requests

REQUEST FORM	*INITIAL VALUE*	*IF NO ARGUMENT*
.bd *F N*	off	-

Embolden font *F* by *N*-1 units. 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 in the vicinity of 10 points. If *N* is missing, the embolden mode is turned off. The mode must still (or again) be in effect when the characters are physically printed. There is no effect in the **nroff** formatter.

REQUEST FORM	*INITIAL VALUE*	*IF NO ARGUMENT*
.bd *S F N*	off	-

Embolden special font when current font is *F*. The characters in the special font will be emboldened whenever the current font is *F*. The mode must still (or again) be in effect when the characters are physically printed. There is no effect in the **nroff** formatter.

REQUEST FORM	*INITIAL VALUE*	*IF NO ARGUMENT*
.fp *N F*	R,I,B,S	ignored

Font position. A font named *F* is mounted on position *N*. It is a fatal error if *F* is not known.

REQUEST FORM	*INITIAL VALUE*	*IF NO ARGUMENT*
.ft *F*	Roman	previous

Change to font *F* (*F* is *x*, *xx*, digit, or *P*). Font *P* means the previous font. For font changes within a line of text, sequences **f***x*, **f**(*xx*, or **f***N* can be used. Relevant parameters are a part of the current environment.

8. Character Size Control Requests

REQUEST FORM	*INITIAL VALUE*	*IF NO ARGUMENT*
.cs *F N M*	off	-

Set constant character space (width) mode on for font *F* (if mounted). The width of every character is assumed to be *N*/36 ems. If *M* is absent, the em is that of the character 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 still (or again) be in effect when the characters are printed. There is no effect in the **nroff** formatter.

REQUEST FORM	*INITIAL VALUE*	*IF NO ARGUMENT*
.ps ±*N*	10 point	previous

Set point size to ±*N*. Any valid positive size value may be requested; if invalid, the next larger valid size will result (maximum of 36). Valid point sizes depend upon the typesetter used. A paired sequence +*N*, -*N* will work because the previous requested value is remembered. For point size changes within a line of text, sequences **s***N* or **s*±*N* can be used. Relevant parameters are a part of the current environment. There is no effect in the **nroff** formatter.

REQUEST FORM	*INITIAL VALUE*	*IF NO ARGUMENT*
.ss *N*	12/36 em	ignored

Set space-character size to *N*/36 ems. This size is the minimum word spacing in adjusted text. Relevant parameters are a part of the current environment. There is no effect in the **nroff** formatter.

9. Page Control Requests

Note: Values separated by ";" are for the **nroff** and **troff/otroff** formatters, respectively.

REQUEST FORM	*INITIAL VALUE*	*IF NO ARGUMENT*
.bp $\pm N$	$N = 1$	-

Begin page. The current page is ejected and a new page is begun. If $\pm N$ is given, the new page number will be $\pm N$. The scale indicator is ignored if not specified in the request. The request causes a break. The use of "'" as the control character (instead of ".") suppresses the break function. The request with no N is inhibited by the **.ns** request.

REQUEST FORM	*INITIAL VALUE*	*IF NO ARGUMENT*
.mk R	none	internal

Mark current vertical place in an internal register (associated with the current diversion level) or in register R, if given. The request is used in conjunction with "return to marked vertical place in current diversion" request (**.rt**). Mode or relevant parameters are associated with current diversion level.

REQUEST FORM	*INITIAL VALUE*	*IF NO ARGUMENT*
.ne N	-	$N = 1V$

Need N vertical spaces. The scale indicator is ignored if not specified in the request.

- If the distance to the next trap position (D) 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 is less than vertical spacing (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. Mode or relevant parameters are associated with current diversion level.

REQUEST FORM	*INITIAL VALUE*	*IF NO ARGUMENT*
.pl ±*N*	11in	11in

Page length set to ±*N*. The internal limitation is about 75 inches in the **troff** formatter and 136 inches in the **nroff** formatter. Current page length is available in the **.p** register. The scale indicator is ignored if not specified in the request.

REQUEST FORM	*INITIAL VALUE*	*IF NO ARGUMENT*
.pn ±*N*	*N* = 1	ignored

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

REQUEST FORM	*INITIAL VALUE*	*IF NO ARGUMENT*
.po ±*N*	0;26/27in	previous

Page offset. The current left margin is set to ±*N*. The scale indicator is ignored if not specified in the request. The **troff** formatter initial value provides about 1 inch of paper margin including the physical typesetter margin of 1/27 inch. In the **otroff** formatter the maximum (line-length) + (page-offset) is about 7.54 inches. The current page offset is available in the **.o** register.

REQUEST FORM	*INITIAL VALUE*	*IF NO ARGUMENT*
.rt ±*N*	none	internal

Return (upward only) to marked vertical place in current diversion. If ±*N* (with respect to place) is given, the vertical place is ±*N* from the top of the page or diversion. If *N* is absent, the vertical place is marked by a previous **.mk**. The **.sp** request may be used in all cases instead of **.rt** by spacing to the absolute place stored in an explicit register; e.g., using the sequence **.mk** *R*....**sp !***R***u**. Mode or relevant parameters are associated with current diversion level. The scale indicator is ignored if not specified in the request.

10. Text Filling, Adjusting, and Centering Requests

Note: Values separated by ";" are for the **nroff** and **troff/otroff** formatters respectively.

REQUEST FORM	*INITIAL VALUE*	*IF NO ARGUMENT*
.ad *N*	adjust	adjust

Adjust. Output lines are adjusted with mode *N*. If the type indicator (*N*) is present, the adjustment type is as follows:

N	*ADJUSTMENT TYPE*
l	adjust left margin only
r	adjust right margin only
c	center
b or n	adjust both margins
absent	unchanged

The adjustment type indicator *N* may also be a number obtained from the **.j** register. If fill mode is not on, adjustment will be deferred. Relevant parameters are a part of the current environment.

REQUEST FORM	*INITIAL VALUE*	*IF NO ARGUMENT*
.br	-	-

Break. Filling of the line currently being collected is stopped and the line is output without adjustment. Text lines beginning with space characters and empty text lines (blank lines) also cause a break.

REQUEST FORM	*INITIAL VALUE*	*IF NO ARGUMENT*
.ce *N*	off	$N = 1$

Center. The next *N* input text lines are centered within the current line-length. 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. The request normally causes a break. Relevant parameters are a part of the current environment.

REQUEST FORM	*INITIAL VALUE*	*IF NO ARGUMENT*
.fi	fill	-

Fill mode. The request causes a break. Subsequent output lines are filled to provide an even right margin. Relevant parameters are a part of the current environment.

REQUEST FORM	*INITIAL VALUE*	*IF NO ARGUMENT*
.na	adjust	-

No adjust. Output line adjusting is not done. Since adjustment is turned off, the right margin will be ragged. Adjustment type for the **.ad** request is not changed. Output line filling still occurs if fill mode is on. Relevant parameters are a part of the current environment.

REQUEST FORM	*INITIAL VALUE*	*IF NO ARGUMENT*
.nf	fill	-

No-fill mode. Subsequent output lines are neither filled nor adjusted. The request normally causes a break. Input text lines are copied directly to output lines without regard for the current line length. Relevant parameters are a part of the current environment.

11. Vertical Spacing Requests

Note: Values separated by ";" are for the **nroff** and **troff/otroff** formatters respectively.

REQUEST FORM	*INITIAL VALUE*	*IF NO ARGUMENT*
.ls *N*	$N = 1$	previous

Line spacing set to $\pm N$. Output N-1 blank lines (*V*s) after each output text line. If the text or previous appended blank line reached a trap position, appended blank lines are omitted. Relevant parameters are a part of the current environment.

REQUEST FORM	*INITIAL VALUE*	*IF NO ARGUMENT*
.ns	space	-

Set no-space mode on. The no-space mode inhibits **.sp** and **.bp** requests without a next page number. It is turned off when a line of output occurs or with the **.rs** request. Mode or relevant parameters are associated with current diversion level.

REQUEST FORM	*INITIAL VALUE*	*IF NO ARGUMENT*
.os	-	-

Output saved vertical space. The request is used to output a block of vertical space requested by an earlier **.sv** request. The no-space mode (**.ns**) has no effect.

REQUEST FORM	*INITIAL VALUE*	*IF NO ARGUMENT*
.rs	-	-

Restore spacing. The no-space mode (**.ns**) is turned off. Mode or relevant parameters are associated with current diversion level.

REQUEST FORM	*INITIAL VALUE*	*IF NO ARGUMENT*
.sp *N*	-	$N = 1V$

Space vertically. The request provides spaces 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. If the no-space mode (**.ns**) is on, no spacing occurs. The scale indicator is ignored if not specified in the request. The request causes a break.

REQUEST FORM	*INITIAL VALUE*	*IF NO ARGUMENT*
.sv *N*	-	*N* = 1*V*

Save a contiguous vertical block of size *N*. If the distance to the next trap is greater than *N*, *N* vertical spaces are output. If the distance to the next trap is less than *N*, no vertical space is immediately output; but *N* is remembered for later output (**.os**). Subsequent **.sv** requests overwrite any still remembered *N*. The no-space mode (**.ns**) has no effect. The scale indicator is ignored if not specified in the request.

REQUEST FORM	*INITIAL VALUE*	*IF NO ARGUMENT*
.vs *N*	1/6in;12pts	previous

Set vertical base-line spacing size *V*. Transient extra vertical spaces are available with **\x'***N***'**. The scale indicator is ignored if not specified in the request. Relevant parameters are a part of the current environment.

REQUEST FORM	*INITIAL VALUE*	*IF NO ARGUMENT*
Blank text line	-	-

This condition causes a break and output of a blank line (just as does **.sp 1**).

12. Line Length and Indenting Requests

REQUEST FORM	*INITIAL VALUE*	*IF NO ARGUMENT*
.in $\pm N$	$N = 0$	previous

Indent. The indent is set to $\pm N$ and prepended to each output line. The scale indicator is ignored if not specified in the request. Relevant parameters are a part of the current environment. The request causes a break.

REQUEST FORM	*INITIAL VALUE*	*IF NO ARGUMENT*
.ll $\pm N$	6.5 in	previous

Line length. The line length is set to $\pm N$. In the **otroff** formatter, the maximum (line-length) + (page-offset) is about 7.54 inches. The scale indicator is ignored if not specified in the request. Relevant parameters are a part of the current environment.

REQUEST FORM	*INITIAL VALUE*	*IF NO ARGUMENT*
.ti $\pm N$	-	ignored

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. The scale indicator is ignored if not specified in the request. Relevant parameters are a part of the current environment. The request causes a break.

13. Macro, String, Diversion, and Trap Requests

REQUEST FORM	*INITIAL VALUE*	*IF NO ARGUMENT*
.am *xx yy*	-	*.yy* = ..

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

REQUEST FORM	*INITIAL VALUE*	*IF NO ARGUMENT*
.as *xx string*	-	ignored

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

REQUEST FORM	*INITIAL VALUE*	*IF NO ARGUMENT*
.ch *xx N*	-	-

Change trap location. Change the trap position for macro *xx* to be *N*. In the absence of *N*, the trap, if any, is removed. The scale indicator is ignored if not specified in the request.

REQUEST FORM	*INITIAL VALUE*	*IF NO ARGUMENT*
.da *xx*	-	end

Divert and append to macro *xx* (append version of the **.di** request). Mode or relevant parameters are associated with current diversion level.

REQUEST FORM	*INITIAL VALUE*	*IF NO ARGUMENT*
.de *xx yy*	-	*.yy* = ..

Define or redefine 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*. The macro *yy* is then 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 "..".

REQUEST FORM	*INITIAL VALUE*	*IF NO ARGUMENT*
.di *xx*	-	end

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. Mode or relevant parameters are associated with current diversion level.

REQUEST FORM	*INITIAL VALUE*	*IF NO ARGUMENT*
.ds *xx string*	-	ignored

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

REQUEST FORM	*INITIAL VALUE*	*IF NO ARGUMENT*
.dt *N xx*	-	off

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. Mode or relevant parameters are associated with current diversion level. The scale indicator is ignored if not specified in the request.

REQUEST FORM	*INITIAL VALUE*	*IF NO ARGUMENT*
.em *xx*	none	none

End macro. Macro *xx* will be invoked when all input has ended. The effect is the same as if the contents of *xx* had been at the end of the last file processed.

REQUEST FORM	*INITIAL VALUE*	*IF NO ARGUMENT*
.it *N xx*	-	off

Input-line-count trap. An input-line-count trap is set to invoke the macro *xx* after *N* lines of text input have been read (control or request lines do not count). Text may be in-line or interpolated by in-line or trap-invoked macros. Relevant parameters are a part of the current environment.

REQUEST FORM	*INITIAL VALUE*	*IF NO ARGUMENT*
.rm *xx*	-	ignored

Remove. A request, macro, or string is removed. The name *xx* is removed from the name list and any related storage space is freed. Subsequent references have no effect.

REQUEST FORM	*INITIAL VALUE*	*IF NO ARGUMENT*
.rn *xx yy*	-	ignored

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

REQUEST FORM	*INITIAL VALUE*	*IF NO ARGUMENT*
.wh *N xx*	-	-

When. A location trap is set to invoke macro *xx* at page position *N*; a negative *N* is interpreted with respect to 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 found trap at *N*, if any, is removed. The scale indicator is ignored if not specified in the request.

14. Number Registers Requests

REQUEST FORM	*INITIAL VALUE*	*IF NO ARGUMENT*
.af *R c*	Arabic	-

Assign format. Format *c* is assigned to register *R*. Available formats are:

c	*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,...,z,aa,ab,...,zz,aaa,...
A	0,A,B,...,Z,AA,AB,...,ZZ,AAA,...

An Arabic format having *N* digits specifies a field width of *N* digits. Read-only registers and width function are always arabic.

REQUEST FORM	*INITIAL VALUE*	*IF NO ARGUMENT*
.nr *R* ±*N M*	-	-

Number register. The number register *R* is assigned the value ±*N* with respect to the previous value, if any. The automatic incrementing value is set to *M*. The number register value (*N*) is ignored if not specified in the request.

REQUEST FORM	*INITIAL VALUE*	*IF NO ARGUMENT*
.rr *R*	-	-

Remove register. The number register *R* is removed. If many registers are being created dynamically, it may be necessary to remove registers that are no longer used in order to recapture internal storage space for newer registers.

15. Tab, Leader, and Field Requests

Note: Values separated by ";" are for the **nroff** and **troff/otroff** formatters respectively.

REQUEST FORM	*INITIAL VALUE*	*IF NO ARGUMENT*
.fc *a b*	off	off

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.

REQUEST FORM	*INITIAL VALUE*	*IF NO ARGUMENT*
.lc *c*	.	none

Leader repetition character becomes *c* or is removed specifying motion. Relevant parameters are a part of the current environment.

REQUEST FORM	*INITIAL VALUE*	*IF NO ARGUMENT*
.ta *Nt...*	8n;0.5 in	none

Set tab stops and types. The adjustment within the tab is as follows:

t	*ADJUSTMENT TYPE*
R	right
C	centering
absent	left

Tab stops for the **troff** formatter are preset every 0.5 inch; Tab stops for the **nroff** formatter are preset every eight nominal character widths. Stop values are separated by spaces, and a value preceded by + is treated as an increment to the previous stop value. Relevant parameters are a part of the current environment. The scale indicator is ignored if not specified in the request.

REQUEST FORM	*INITIAL VALUE*	*IF NO ARGUMENT*
.tc *c*	none	none

Tab repetition character becomes *c* or is removed specifying motion. Relevant parameters are a part of the current environment.

16. Input/Output and Translation Requests

Note: Values separated by ";" are for the **nroff** and **troff/otroff** formatters respectively.

REQUEST FORM	*INITIAL VALUE*	*IF NO ARGUMENT*
.cc *c*	.	.

Set control character to **c** or reset to ".". Relevant parameters are a part of the current environment.

REQUEST FORM	*INITIAL VALUE*	*IF NO ARGUMENT*
.cu *N*	off	*N* = 1

Continuous underline in the **nroff** formatter. A variant of **.ul** that causes every character to be underlined. Identical to **.ul** in the **troff** formatter. Relevant parameters are a part of the current environment.

REQUEST FORM	*INITIAL VALUE*	*IF NO ARGUMENT*
.c2 *c*	'	'

Set no-break control character to **c** or reset to "'". Relevant parameters are a part of the current environment.

REQUEST FORM	*INITIAL VALUE*	*IF NO ARGUMENT*
.ec *c*	\	\

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

REQUEST FORM	*INITIAL VALUE*	*IF NO ARGUMENT*
.eo	on	-

Turn escape character mechanism off.

REQUEST FORM	*INITIAL VALUE*	*IF NO ARGUMENT*
.lg *N*	off;on	on

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

REQUEST FORM	*INITIAL VALUE*	*IF NO ARGUMENT*
.tr *abcd...*	none	-

Translate *a* into *b*, *c* into *d*, etc. on output. 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. Initially there are no translate values.

REQUEST FORM	*INITIAL VALUE*	*IF NO ARGUMENT*
.uf *F*	Italic	Italic

Underline font set to *F* (to be switched to by **.ul**). In the **nroff** formatter *F* may not be on position 1 (initially Times Roman).

REQUEST FORM	*INITIAL VALUE*	*IF NO ARGUMENT*
.ul *N*	off	*N* = 1

Underline in the **nroff** formatter (italicize in **troff**) the next *N* input text lines. 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** is affected by the font change but does not decrement *N*. If *N* is greater than 1, there is the risk that a trap interpolated macro may provide text lines within the span, which environment switching can prevent. Relevant parameters are a part of the current environment.

17. Hyphenation Requests

REQUEST FORM	*INITIAL VALUE*	*IF NO ARGUMENT*
.hc *c*	\%	\%

Hyphenation character. Hyphenation indicator character is set to *c* or to the default "\%". The indicator does not appear in the output. Relevant parameters are a part of the current environment.

REQUEST FORM	*INITIAL VALUE*	*IF NO ARGUMENT*
.hw *word1...*	-	ignored

Exception words. Hyphenation points in words are specified 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. Space available is small — about 128 characters.

REQUEST FORM	*INITIAL VALUE*	*IF NO ARGUMENT*
.hy *N*	off,$N=0$	on,$N=1$

Hyphenate. Automatic hyphenation is turned on for $N \geq 1$ or off for $N=0$. If $N=2$, last lines (ones that will cause a trap) are not hyphenated. For $N=4$ the last two characters of a word are not divided. For $N=8$ the first two characters of a word are not divided. These values are additive; i.e., $N=14$ invokes all three restrictions. Relevant parameters are a part of the current environment.

REQUEST FORM	*INITIAL VALUE*	*IF NO ARGUMENT*
.nh	no hyphen	-

No hyphenation. Automatic hyphenation is turned off. Relevant parameters are a part of the current environment.

18. Three-Part Title Requests

REQUEST FORM	*INITIAL VALUE*	*IF NO ARGUMENT*
.lt ±*N*	6.5 in	previous

Length of title set to ±*N*. Line length and title length are independent. Indents do not apply to titles; page offsets do. Relevant parameters are a part of the current environment. The scale indicator is ignored if not specified in the request.

REQUEST FORM	*INITIAL VALUE*	*IF NO ARGUMENT*
.pc *c*	%	off

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

REQUEST FORM	*INITIAL VALUE*	*IF NO ARGUMENT*
.tl'*l*'*c*'*r*'	-	-

Three-part title. The strings *l*, *c*, and *r* 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 having the format assigned to register %. Any character may be used as the string delimiter.

19. Output Line Numbering Requests

REQUEST FORM	*INITIAL VALUE*	*IF NO ARGUMENT*
.nm $\pm N\ M\ S\ I$	-	off

Line number mode. If $\pm N$ is given, line numbering is turned on, and the next output line is numbered $\pm N$. 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, and the next line number is preserved for possible further use in number register **ln**. Relevant parameters are a part of the current environment.

REQUEST FORM	*INITIAL VALUE*	*IF NO ARGUMENT*
.nn N	-	$N = 1$

Next N lines are not numbered. Relevant parameters are a part of the current environment.

20. Conditional Acceptance Requests

REQUEST FORM	*INITIAL VALUE*	*IF NO ARGUMENT*

.el *anything*

The "else" portion of "if-else".

REQUEST FORM	*INITIAL VALUE*	*IF NO ARGUMENT*

.ie *c anything*

The "if" portion of "if-else". The *c* can be any of the forms acceptable with the **.if** request.

REQUEST FORM	*INITIAL VALUE*	*IF NO ARGUMENT*

.if *c anything*

If condition *c* true, accept *anything* as input; for multiline case, use \{*anything*\}. The scale indicator is ignored if not specified in the request.

REQUEST FORM	*INITIAL VALUE*	*IF NO ARGUMENT*

.if *!c anything*

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

REQUEST FORM	*INITIAL VALUE*	*IF NO ARGUMENT*

.if *N anything*

If expression $N > 0$, accept *anything*. The scale indicator is ignored if not specified in the request.

REQUEST FORM	*INITIAL VALUE*	*IF NO ARGUMENT*

.if *!N anything*

If expression $N \leq 0$ accept *anything*. The scale indicator is ignored if not specified in the request.

REQUEST FORM	*INITIAL VALUE*	*IF NO ARGUMENT*

.if *'s1' s2' anything*

If string *s1* is identical to string *s2*, accept *anything*.

REQUEST FORM	*INITIAL VALUE*	*IF NO ARGUMENT*

.if *!'s1' s2' anything*

If string *s1* is not identical to string *s2*, accept *anything*.

21. Environment Switching Request

REQUEST FORM	*INITIAL VALUE*	*IF NO ARGUMENT*
.ev *N*	$N = 0$	previous

Environment switched to 0, 1, or 2. Switching is done in pushdown fashion so that restoring a previous environment must be done with **.ev** rather than specific reference.

22. Insertions From Standard Input Requests

REQUEST FORM	*INITIAL VALUE*	*IF NO ARGUMENT*
.ex	-	-

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

REQUEST FORM	*INITIAL VALUE*	*IF NO ARGUMENT*
.rd *prompt*	prompt=BEL	-

Read insertion from the standard input until two newline characters in a row are found. If standard input is the user keyboard, a *prompt* (or a BEL) is written onto the user terminal. The request behaves like a macro; arguments may be placed after *prompt*.

23. Input/Output File Switching Requests

REQUEST FORM	*INITIAL VALUE*	*IF NO ARGUMENT*
.cf *filename*	-	-

Copy file. This request copies the contents of filename into the **troff** output file at this point, uninterpreted. Havoc ensues unless the motions in the file restore current horizontal and vertical position. (Not supported in **otroff**)

REQUEST FORM	*INITIAL VALUE*	*IF NO ARGUMENT*
.nx *filename*	end-of-file	-

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

REQUEST FORM	*INITIAL VALUE*	*IF NO ARGUMENT*
.pi *program*	-	-

Pipe output to *program* (**nroff** and **troff** formatters only, not **otroff**). This request must occur before any printing occurs. No arguments are transmitted to *program.*

REQUEST FORM	*INITIAL VALUE*	*IF NO ARGUMENT*
.so *filename*	-	-

Switch source file (pushdown). The top input level (file reading) is switched to *filename.* Contents are interpolated at the point the request is encountered. When the new file ends, input is again taken from the original file. The **.so** requests may be nested.

24. Miscellaneous Requests

REQUEST FORM	*INITIAL VALUE*	*IF NO ARGUMENT*
.co	-	-

Specify the point in the macro file at which compaction ends. When **-k***name* is called on the command line, all lines in the file *name* before the **.co** request will be compacted (**otroff** only).

REQUEST FORM	*INITIAL VALUE*	*IF NO ARGUMENT*
.fl	-	-

Flush output buffer. Used in interactive debugging to force output. The request causes a break.

REQUEST FORM	*INITIAL VALUE*	*IF NO ARGUMENT*
.ig *yy*	-	*.yy* = ..

Ignore input lines until call of *yy*. This request behaves like the **.de** request except that the input is discarded. The input is read in *copy* mode, and any automatically incremented registers will be affected.

REQUEST FORM	*INITIAL VALUE*	*IF NO ARGUMENT*
.mc *c N*	-	off

Sets margin character *c* and separation *N*. Specifies that a margin character *c* appear a distance *N* to the right of the right margin after each nonempty text line (except those produced by **.tl**). If the output line is too long (as can happen in no-fill 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 the **nroff** formatter and 1 em in **troff**. Relevant parameters are a part of the current environment. The scale indicator is ignored if not specified in the request.

REQUEST FORM	*INITIAL VALUE*	*IF NO ARGUMENT*
.pm *t*	-	all

Print macros. The names and sizes of all defined macros and strings are printed on the user terminal. If *t* is given, only the total of the sizes is printed. Sizes are given in blocks of 128 characters.

REQUEST FORM	*INITIAL VALUE*	*IF NO ARGUMENT*
.sy *cmd args*	-	-

The UNIX system command *cmd* is executed. Its output is not captured. The standard input for *cmd* is closed. (Not supported in **otroff**)

REQUEST FORM	*INITIAL VALUE*	*IF NO ARGUMENT*
.tm *string*	-	newline

Print *string* on terminal (UNIX operating system standard message output). After skipping initial blanks, string (rest of the line) is read in *copy* mode and written on the user terminal.

25. Output and Error Messages Request

REQUEST FORM	*INITIAL VALUE*	*IF NO ARGUMENT*
.ab *text*	-	-

Prints *text* on the message output and terminates without further processing. If *text* is missing, "User Abort." is printed. This request does not cause a break. The output buffer is flushed.

Chapter 4

DEVICE-INDEPENDENT TROFF

PAGE

1. Introduction 4-1
2. Incompatibilities 4-1
3. New Features 4-4
4. Other Important Changes 4-6
5. Caveats 4-6
6. Supported Devices 4-7
7. Parameters of the APS-5 Phototypesetter 4-8

Chapter 4

DEVICE-INDEPENDENT TROFF

1. Introduction

Troff (but not **nroff**) has been rewritten to support phototypesetters other than the Wang C/A/T. This new version of the formatter has been named **troff** and the older version (Wang C/A/T only) has been renamed **otroff** for old troff. Thus, parameters which were previously constants dictated by the physical constraints of the C/A/T can now be varied to satisfy the constraints of other phototypesetters. Example of these parameters are the device resolution (for the C/A/T and **otroff**, 1/432 inch horizontally, and 1/144 inch vertically), and the number of fonts that can be used in a single run (four for the C/A/T). Section 6 describes these parameters and some of the available fonts for supported phototypesetting devices.

The new **troff** accepts the same language as **otroff** (known exceptions are noted below in Part 2). However, its output, instead of being machine codes for the C/A/T phototypesetter, consists of a device-independent American Standard Code for Information Interchange (ASCII) language describing where on the page characters are to be placed by the phototypesetter. This output has been tailored to the resolution and font descriptions of a particular phototypesetter, but otherwise independent of any particular device. To produce the desired phototypeset pages, this language must be translated by another program (called a postprocessor) into the machine codes needed to run that particular phototypesetter. For the output to look optimal, the postprocessor should support the same device as the description tables used earlier by **troff**, but this is not necessary.

2. Incompatibilities

Aside from the fact that the phototypeset output generated by the new **troff** will look slightly different (hopefully better for devices other than the C/A/T), there are certain known incompatibilities from **otroff** to the new **troff**.

2.1 Unsupported Features

Compacted macros are no longer supported by **troff**. This includes the request **.co**, and the command line options **−c** and **−k**.

The constant-width preprocessor **cw** is no longer needed, or supported. To run without **cw**, add the following **troff** requests to your definition of the **.CW** macro:

```
.ft CW
.br
```

and to the definition of the **.CN** macro:

```
.br
.ft
```

Also, all uses of the **.CD**, **.CP** and **.PC** macros and **cw** delimiters must be replaced by their equivalents (using **\f(CW** and **\fP**). Dependence on the transparent mode feature of **cw** must be removed. For users who are unwilling to give up their use of **cw**, there is an unsupported revision of **cw** which should look identical to the old **cw** except for the default font position of the CW font.

2.2 Deleted Options

−p	The **−p** option is no longer supported.
−g	The **−g** option is no longer necessary.
−c, −k	The options **−c** and **−k** have been eliminated, along with compacted macros.
−w, −b	The **−w** and **−b** options have migrated to the postprocessor.

2.3 Deleted Requests

.! — The **.!** request no longer exists. Instead, use **.sy** (see Part 3.2) which does not interpolate its output into the input of **troff**. To capture the output from a **.sy** request, redirect it into a temporary file, perhaps using the new number register $$ (see Part 3.5).

.fz — The **.fz** request to force a size for a particular font no longer exists.

.co — The **.co** request for compacting macros no longer exists.

2.4 Deleted Escape Sequences and Number Registers

\j — The escape sequence **\j** to mark horizontal position on the output line no longer exists.

hp — The number register **\n(hp** (horizontal position on input line) no longer exists.

2.5 Changed Character Names in The CW Font

\(dg — The special character name **\(dg** no longer represents the control-shift indicator in the CW font. It represents the dagger on all fonts. For the control-shift indicator, use the new special character **\(cs** (see Part 3.6).

\(sq — The special character name **\(sq** no longer represents the visible space indicator in the CW font. It represents the square on all fonts. For the visible space indicator, use the new special character **\(vs** (see Part 3.6).

3. New Features

Aside from its device-independence and the loosening of restrictions on fonts, the new **troff** also supplies the following new features:

3.1 New Options

−T*name* — The **−T** option may be used to specify the output device. The default output device is defined locally.

−F*dir* — The **−F** option causes font information accessed from the directory dir/*dev***name** instead of the default /usr/lib/font/dev**name** (where **name** is the default output device).

3.2 New Requests

.cf *file* — The **.cf** request copies the contents of *file* into the **troff** output file at this point, uninterpreted. Havoc ensues unless the motions in the file restore current horizontal and vertical position.

.sy *cmd args* — The UNIX command *cmd* is executed. Its output is not captured anywhere. The standard input for *cmd* is closed.

.pi *cmd* — As in **nroff**, the **.pi** request causes the output of **troff** to be piped into *cmd* instead of appearing on the standard output.

3.3 Modified Requests and Escape Sequences

.ft *F*, **\f***F* — The **.ft** request and the **\f** escape sequence cause the font *F* to be loaded on font position 0 (which is in all other ways inaccessible) if the font exists and is not currently mounted by default or by a **.fp** request. The font must be still or again in font position 0 when the line is printed.

3.4 New Escape Sequences

\D'l *dh dv'* Draw a line from the current position by *dh,dv.*

\D'c *d'* Draw a circle of diameter *d* with left side at current position.

\D'e *d1 d2'* Draw an ellipse of diameters *d1* and *d2* with left side at current position.

\D'a *dh1 dv1 dh2 dv2'*
Draw a counterclockwise arc from current position to *dh1+dh2, dv1+dv2,* with center at *dh1,dv1* from current postion.

\D'~ *dh1 dv1 dh2 dv2 ...'*
Draw a B-spline from current position by *dh1,dv1,* then by *dh2,dv2,* then ... .

\H'*n'* Character heights are set to *n* points, without changing widths. A height of the form ±*n* is an increment on the current point size; a height of zero restores the height to the point size.

\S'*n'* Output is slanted *n* degrees. *n* may be negative. If *n* is zero, slant mode is turned off.

3.5 New Predefined Number Registers and Strings

$$ Read-only. Contains the process-id of the **troff** process. This is useful for temporary file names as in

```
.sy ... >\n($$
```

.T	Contains the name of the **troff** output device, for example, *aps*. This is a string, not a number register, so it is accessed as ***(.T**.

3.6 New Special Character Names

\(cs	Represents the control-shift indicator and is available with any font.
\(vs	Represents the visible space indicator and is available with any font.

4. Other Important Changes

Transparent mode with **\!** has been fixed so that undiverted transparent output actually appears in the output.

5. Caveats

Users must remember that any changes to fonts made using **\f**, **.ft** or **.fp** must still be in effect at the time of actual output of the unit of text, be it a line of output or a diversion. There is a known problem when multiple **\f** escape sequences or multiple **.fp** requests for the same font position occur in too short a sequence. This can cause either incorrect fonts or incorrect spacings of characters in the correct font.

The escape sequence **\s***n* (point size) does not work for *n* larger than 36. Use **.ps** for point sizes larger than 36.

6. Supported Devices

6.1 Phototypesetters

The machine parameters for each supported phototypesetter are contained in a description table which is used by the new **troff** as well as by the postprocessor for that phototypesetter. This description table sets parameters for legal point sizes, default font settings, machine resolution, and legitimate special names for non-ASCII characters.

For phototypesetters other than the Wang C/A/T, it is no longer necessary to mount fonts at the beginning of a document. Fonts can be mounted and remounted at any time, subject to the limitation that they must be still mounted when the actual output is done. A font can be accessed with the **\f** escape sequence or **.ft** request even if that font is not mounted (which is equivalent to mounting it on the "imaginary" font position 0). There is no limit to the number of fonts used in a document.

Currently, only one phototypesetter is supported, which is the **Autologic APS-5**. It is referenced by the **troff** option **−Taps**, and has a postprocessor named **daps**. A table of its relevant parameters is found in Part 7.

6.2 Simulator Postprocessors

In addition to postprocessors for phototypesetters, there are also postprocessors for various devices which can simulate (to a greater or lesser extent) the output of a phototypesetter. These postprocessors can be used with the output from **troff** which has been prepared for any one of a number of different phototypesetters. In other words, these postprocessors do not require their own **−T** option to **troff**. Currently supported devices of this type include postprocessors for the Xerox 9700 laser printer (**dx9700**) and the Canon Imagen Imprint-10 laser printer (**di10**).

7. Parameters of the APS-5 Phototypesetter

RESOLUTION: 723 units per inch.

LEGAL POINT SIZES:

3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
22 24 26 28 30 32 34 36 40 44 48

DEFAULT FONT SETTINGS:

POSITION:	1	2	3	4	5	6	7	8
FONT:	R	I	B	H	CW	S	S1	GR

SLANT: On the APS-5, all type set in slant mode is set at an angle of approximately 14 degrees, regardless of the value of *n* (in \S'*n*'). If *n* is positive, slant will be 14 degrees; if *n* is negative, slant will be reverse 14 degrees.

HEIGHT: On the APS-5, height can be squashed down to the minimum allowable size, but can only be stretched to about 1.5 times its original size.

Refer to Figure 4-1 for a list of available font styles with the APS-5 phototypesetter.

NAME	*FONT*
R	Times Roman
I	*Times Italic*
B	**Times Bold**
BI	***Times Bold Italic***
H	Helvetica Regular
HI	*Helvetica Italic*
HB	**Helvetica Black**
PA	Palatino Regular
PI	*Palatino Italic*
PB	**Palatino Bold**
CE	Century Expanded
CI	***Century Italic***
SM	Stymie Medium
TB	**Techno Bold**
C	News Gothic Condensed
CW	Constant Width
CT	Courier Typewriter
GS	German Script
SC	*Regular Script*

Figure 4-1. Available Font Styles for the APS-5

Chapter 5

SROFF TUTORIAL GUIDE

		PAGE
1.	Introduction	5-1
2.	Heads and Feet	5-3
3.	Line Breaks	5-3
4.	Finding Troubles	5-6
5.	Pages, Widows, Figures	5-6
6.	Underlining and Boldface	5-9
7.	Tabulation	5-9
8.	Footnotes	5-10
9.	Literal	5-11
10.	Translation	5-12
11.	Hyphenation	5-13
12.	Source Switching	5-14
13.	Numbering	5-14
14.	Dates, Number Formats	5-16
15.	Indexing, Cross-referencing	5-16
16.	Page Offset, Line Numbering	5-18
17.	Text Registers	5-18
18.	Parameters	5-20
19.	Use of the Ignore Request	5-21
20.	Merge Patterns	5-22
21.	Multiple Columns	5-23
22.	Conclusion	5-23

Chapter 5

SROFF TUTORIAL GUIDE

1. Introduction

This chapter is a tutorial introduction to the **sroff** text formatter. The **sroff** formatter is designed to produce output that will be printed on a typewriter-like printer or higher quality laser printer. The formatting requests **sroff** understands are similar to, but not compatible with, **nroff/troff** requests. Since the **sroff** formatter is less complex than the **nroff** formatter, it is much faster and therefore provides an advantage for documents that do not require high-quality phototypeset output.

Consider the example of **sroff** output on the next page.

SROFF TEXT FORMATTER

Tutorial Guide

The title lines, "SROFF TEXT FORMATTER" and "Tutorial Guide," and this paragraph were set up by the input that follows. One extra line (.sp) was spaced down, and then each title line was centered (.ce). An extra line was put between the two title lines. The paragraph begins after another spacing line and is double spaced (.ds). The paragraph was typed without attention to lines; Sroff took care of filling them. Each sentence begins on a separate line to make editing easier.

```
.ce
SROFF TEXT FORMATTER
.sp
.ce
Tutorial Guide
.sp
.ds
The title lines, "SROFF TEXT FORMATTER" and
"Tutorial Guide," and this first paragraph
were set up by the input that follows.
One extra line (.sp) was spaced down,
and then each title
line was centered (.ce).
An extra line was put between the two title lines.
The paragraph begins after another spacing line
and is double spaced (.ds).
The paragraph was typed without attention to lines;
Sroff took care of filling them.
Each sentence begins on a separate line to make
editing easier.
```

2. Heads and Feet

A footing line at the bottom of each page can be specified by the request .ef for even-numbered pages and .of for odd-numbered pages. Similar requests, .eh and .oh, set up the headings. Quote marks split up the foot into parts for the left and right side of the page. Between the two middle quote marks is stuff for the middle of the line. The % mark shows where the page number is to go.
The following input:

```
.ef 'Your Company'- T-% -' ^(amon) 19^(year)'
.of ' ^(amon) 19^(year)'- T-% -'Your Company'
```

will produce at the bottom of even pages

```
Your Company          - T-(Page #) -          October 1983
```

and at the bottom of odd pages

```
October 1983          - T-(Page #) -          Your Company
```

If you want the same heading on all pages, then a single .he request will do. To set up a standard style of page numbering for memoranda use

```
.he ''- % -''
```

3. Line Breaks

In ordinary paragraph text, Sroff takes care of filling lines with words, but there are places where one wants to control exactly where the line ends. The titles in the example of Part 1, " SROFF TEXT FORMATTER" and " Tutorial Guide," should each have a line to itself, and each paragraph is expected to begin on a new line, not just run on to the paragraph before. Certain requests, among them .ce and .sp, cause line breaks; others are explained in the Chapter 6.

Sometimes a break on every line is needed. One could place a simple break (.br) request between every pair of lines, as in X. J. Kennedy's first stanza below, but it is easier to shut off all filling of lines by .nf (no fill) as in the second. The .fi request turns filling on again.

```
In a prominent bar in Secaucus one day
.br
Rose a lady in skunk with a top-heavy sway,
.br
Raised a knobby red finger--all turned from their beer--
.br
While with eyes bright as snowcrust she sang high
and clear:
.sp
.nf
"Now who of you'd think from an eyeload of me
That I once was a lady as proud as could be?
Oh, I'd never sit down by a tumbledown drunk
If it wasn't, my dears, for the high cost of junk."
.fi
```

Normally all paragraphs are justified, that is, enough extra spaces are put into every line to align the right margin exactly (as shown in the example in Part 1). Ordinary typewriter style with an uneven right margin results from a no justification (.nj) request.

Consider the following example.

```
.nj
.ti 5

        This paragraph is not justified.  It was

surrounded by the no justification (.nj) and

justification (.ju) requests that you see.  The first

line was indented 5 spaces by a temporary indent (.ti)

request that holds for exactly one line.

.ju
```

Lines may still be filled even with justification off, but justification is never done when nofill (.nf) is in effect. Now for a different look.

```
.ss
.in5
.ll55
.ti+5
          This paragraph was single spaced
     (.ss), indented (.in) 5 spaces and the
     right margin was shortened 5 by setting
     the line length (.ll). (The line length
     includes the indentation.) Indentation
     and line length may also be changed by
     addition and subtraction as in .ti+5.
     The indentation and line length were set
     back to normal after the paragraph.
.sp
.ll65
.in0
.ds
```

4. Finding Troubles

Because all the requests vanish when you Sroff something, it is sometimes tricky to figure out the trouble when things go wrong. The following paragraph shows how it can be done.

```
The requests that made this paragraph   278 .pr.ps.uf.ul
were caused to be printed in the right  279
margin beyond the 50-character lines by 280
.pr 64.  To help in locating text in the 281
original input, the sequence numbers of 282
the input lines were printed at column 61 283
by .ps 60.  To see the exact division of 284
the original input lines, the first char- 285
acter of each was underlined by .uf.  All 286
this was turned off at the end by .pr 0, 287
.ps 0, and .nu.                         288 .pr.ps.nu.sp
```

5. Pages, Widows, Figures

These requests cause a new page to begin:

.bp begins a page

.pa n begins a page and sets its number to n

Ordinarily Sroff begins a new page only when the page before is filled up. It can very easily create "widows," isolated lines that really belong with the page before or the page after. Typical cases are a subhead that falls at the bottom of a page, or the last couple of words of a paragraph that fall at the top. When a widow does crop

up, you can fix it with a need (.ne) request. In the next example, .ne 4 says no more lines should be put on the current page unless four lines can be put there, thus assuring that at least the first two lines of the single-spaced paragraph go along with the subhead.

```
.ss
.ne 4
.ce
Control of Radioactive Pollution
.sp
No really effective therapy is known for preventing or
curing the harmful effects of internal contamination
by radioactive nuclides.
```

Need requests come in handy as well for guaranteeing that a table or a set of equations not be split across pages.

Widows that appear at the top of a page while really belonging on the previous page pose a more difficult problem. One could put a .ne 2 request somewhere close to the end of a paragraph to force the last two lines to appear on the same page (.ne doesn't cause a line break), but it is generally unknown where new output lines will begin when you are in fill mode. Instead an automatic widow suppression mechanism is available. If you are in single space fill mode and are not centering lines, then by default any .sp request (empty lines are the same as .sp) behaves like this sequence:

```
.m4 -1
.sp
.m4 +1
.ne 2
```

The bottom margin is reduced by one, making space on the page for one extra line of text, the .sp causes a line break which flushes the last line or the paragraph onto the page, the bottom margin is reset to its original value, and the .ne 2 request causes a new page to begin if there is not space on the current page for at least two lines.

This has the effect of automatically suppressing all one-line widows in fill mode. It also has the effect of pushing one-line paragraphs

that would have fit exactly on the bottom of the page to the top of the next page.

It is possible to do something about leading widows via

```
.ws 0  perform no widow suppression
.ws 1  (default) suppress 1-line leading and trailing widows
.ws 2  suppress 2-line leading and 1-line trailing widows
.ws 3  suppress 3-line leading and 1-line trailing widows
...
```

but multi-line trailing widows are still a problem with no general solution. Sroff does the obvious thing in double space or multi-space mode; .ws operates in terms of the line spacing. Obvious boundary condition errors are avoided, such as the bottom margin being less than the line spacing.

A need request may be used to set aside space for a figure. Both these examples leave room for 12 lines (two inches):

```
.ss                .ds
.ne 12             .ne 6
.sp 12             .sp 12
```

An obvious trouble with .ne for this purpose is its dependence on line spacing; another is that it can leave more space than you want. In these examples there could be as much as 11 lines empty at the bottom of one page before the 12-line figure at the top of the next. A .lv (leave) request does a better job for figures. It leaves space on the current page if there is room, but remembers the request until the next page if there isn't. Thus

```
.lv 12
```

would set aside space as well as the previous examples, but would not insist that the space come right where the request appeared.

6. Underlining and Boldface

The best way to do underlining is to use the .ul n request, which causes the *next n lines of input* to be underlined regardless of line filling. Putting underlines into filled text by backspacing and overstriking is a ticklish procedure that can give troublesome effects.

Punctuation characters are not touched by .ul. To underline punctuation you can use the .us request, which underlines even the space character. It is best to use this request in nofill mode if blanks are included in the lines to be underscored.

The .bf n request causes the next n input lines to appear in boldface. Since most output devices don't have this capability, it is usually simulated by n successive overstrikes, where n is settable by .bo n. Boldface can also be achieved by backspacing over the text and retyping the overstrikes yourself. In any case, always observe the cardinal rule for typing filled text: *Never backspace across a space.*

7. Tabulation

Tab stops may be set in Sroff as on a typewriter, but in addition to creating columns of data aligned along their left edges, tabs may also create columns aligned along their right edges, or columns aligned by centering. The alignment is done during *input* before any other formatting. Thus tabs work best in nofill mode, but filling of the rightmost column is possible.

Tabs are set by a .ta request, which shows in what columns tabs are to be set, and actuated by the presence of a *tab character.* Tabs for left-aligned columns of data are set by an " L" at the leftmost position of a field, for right-aligned fields by an " R" at the rightmost position, and for centered fields by a " C" at the middle position. It is not necessary for your terminal tab stops to be correctly positioned; Sroff replaces all tabs by an appropriate number of spaces.

In the example below, we wish to right-align the left sides of equations, (left-) align the equals signs, and right-align the equation numbers. To make the tab character visible we used .tc :. Notice that there must be a tab character wherever spaces are to be

inserted, including the space at the left margin before the first tabbed field.

```
.nf
.in5
.tc:
.ta 10R 12L 40R
:sin x:= x - x**3/3! + x**5/5! - ...:(9)
:sinh x:= x + x**3/3! + x**5/5! + ...:(10)
```

Here is the result. The pointers are at 10, 20, 30, etc.

```
 sin x   = x - x**3/3! + x**5/5! - ...    (9)
sinh x   = x + x**3/3! + x**5/5! + ...   (10)

      ↑            ↑            ↑            ↑
```

Tabs can be set by addition and subtraction to save the trouble of calculating positions when all you know is column widths. For example, to divide the page into 8-character columns write

```
.ta 1 +8 +8 +8 +8 +8 +8
```

This one sets left- and right-justified stops at 60 and 54.

```
.ta 60L -6R
```

8. Footnotes

Footnotes can be placed anywhere in the input; Sroff collects them and places them at the bottom of the page. A footnote is set off by .fn and .en. With only minor exceptions, .fn, .en and everything in between are completely parenthetical to the surrounding text. Indentation, line length, filling, justification, single- and double-spacing, line breaks, tab settings, etc. are all independently handled in footnotes and their settings are remembered from footnote to footnote. You may want to include a .ne request to assure that a

footnote and the reference* to it have room on the same page, as was done here.

```
.fs'----------'''
You may want to include a .ne request to
assure that a footnote and the
.ne 5
reference*
.fn
* Otherwise it is possible for some or all of the
footnote to spill over to the next page, or for the
line containing the reference to get pushed over.
.en
to it have room on the same page, as was done here.
```

Because .ne counts in units of the current line spacing (in this case single spacing), 5 lines is enough to take care of the reference, the one-line separator between body text and footnotes, and the three-line footnote. The .fs request sets the footnote separator line in the same way that .he and .fo set head and foot titles.

9. Literal

Occasionally you may want to type a line beginning with a period, which looks like a Sroff request. A literal (.li) request put right before the line will prevent Sroff from mistakenly trying to interpret it and cause the line to be accepted as ordinary text.

```
.li
.sp this couldn't be printed without .li
```

If the occurrences are many, you may find it more convenient to redefine the control character, thus liberating "." from its special meaning when it begins a line:

```
.cc !
.sp leading . isn't special now
!he 'but leading ! is'
```

* Otherwise it is possible for some or all of the footnote to spill over to the next page, or for the line containing the reference to get pushed over.

10. Translation

Translation can be used to overcome some unpleasing results of justification and filling, such as

```
(1) extra spaces between a hanging paragraph
      number and the rest of the first line
      of the paragraph
(2) unwanted break inside a formula: 1 + x +
      x**2/2! + x**3/3! + ...
```

To fix up such annoyances, you may designate an otherwise unused character, say $, to be translated (.tr) into a space. Though it will print as a space, Sroff does not treat it as such, and so won't pad or break lines there.

```
.ll 48
.in 10
.tr $
.ti-4
(1)$extra spaces between a hanging paragraph
number and the rest of the first line of the paragraph
.ti-4
(2)$unwanted break inside a formula:
1$+$x$+$x**2/2!$+$x**3/3!$+$...
```

The prettied-up result follows.

```
(1) extra spaces between a hanging
    paragraph number and the rest of the
    first line of the paragraph
(2) unwanted break inside a formula:
    1 + x + x**2/2! + x**3/3! + ...
```

11. Hyphenation

To improve the fit, Sroff ordinarily tries to hyphenate at the end of filled lines. Errors of commission and omission sometimes happen, as here where the column width has been set to 6 letters.

```
ces-
spool
cour-
thouse
tetrabromomethane
```

Hyphenation mode (.hy) requests control the boldness with which Sroff inserts hyphens. Words are never broken under .hy 0; words are quite frequently broken under .hy 3. If you find automatic hyphenation generally displeasing, the best setting will probably be .hy 1. In that mode already hyphenated words, like "run-of-the-mine," may be broken, but nothing else will.

Other places may be marked as candidates for hyphenation by means of a *hyphenation character,* an otherwise unused character, designated in a .hc request. Setting .hc \ and supplying syllabification for the unfortunate "tetra\bromo\meth\ane," we can arrange for satisfactory hyphenation with almost any column width. Here the widths are 6, 12 and 18.

```
width = 6

tetra-
bromo-
meth-
ane

width = 12

tetrabromo-
methane

width = 18

tetrabromomethane
```

Words containing - signs or hyphenation characters are not split at other places. This convention permits a trick for completely avoiding hyphenation of a selected word: put a hyphen character at the beginning or end of the word. Here we arrange that a proper name not be broken and also (by .tr) that initials be kept with it.

```
.tr $
.hc J.$Q.$Public\ can't be fooled
```

The additional .hp request controls hyphenation in words spelled with all CAPITAL letters. Frequently, these are acronyms (like FORTRAN) not suitable for hyphenation.

12. Source Switching

The .so request causes Sroff to switch temporarily to another file for input, then return to the current file when the other is exhausted. It is particularly useful for long documents that have been edited in pieces. For example if the pieces are in files chap1, chap2, chap3, you might prepare a very short file called book, containing only

```
.so /chap1
.so /chap2
.so /chap3
```

then do the whole works by the simple command " sroff book" .

13. Numbering

Sroff can help keep book on numbers of paragraphs, equations, etc. Pick any letter, say P, to name a number, and assign it a numeric value with a .an request. Thus to make P become 1 write

```
.an P 1
```

or to increase P by 1 write

```
.an P +1
```

To place the number in text (or in a request), mention its name right after an *insertion character* of your choosing. The .ic request specifies the insertion character.

Paragraphs of this tutorial guide could have been numbered by a quantity P, which would be bumped up 1 before each paragraph:

```
.ic ^
.an P 0
   ... text ...
.an P +1
.ul
^P. Numbering.
   ... text ...
```

Names of numbers are called *registers.* Digits may be used as well as letters. Names up to eight letters or digits long can be used, but they must be enclosed in parentheses. For example we might have called the paragraph number (par) and used it this way:

```
.an (par) +1
^(par). Numbering.
```

Once declared by an .ic request, the insertion character will disappear from all input text and request lines up through the next .ic. When the character right after a disappearing insertion character is not the name of a register, that character goes through unchanged—even if it is the insertion character. It may be instructive to puzzle out the reason for the behavior of this sequence:

```
.ic ^     make ^ the insertion character
.ic ^     turn off insertion character
.ic ^     make ^ the insertion character
.ic ^^    make ^ the insertion character
```

14. Dates, Number Formats

You can get today's date into a document being Sroffed by inserting the registers (year), (mon), and (day), which are set automatically by Sroff. Thus

```
The date is ^(mon)/^(day)/^(year)
```

turns into

```
The date is 10/27/83
```

More pleasant is

```
The date is ^(amon) ^(day), 19^(year)
```

which becomes

```
The date is October 27, 1983
```

The time is similarly available in registers (hour), (min), and (sec). If you are addicted to military-style time, e.g. 0800 hours, where padding with zeroes may be required, you can get it by assigning a special *format* to the registers with .af.

```
.af (hour) 01
.af (min) 01
```

The format, in this case 01, shows how the number 1 is to print. Usually the format is "1". The formats "i" and "I" get Roman numerals. Formats "a" and "A" get "theater-row" numbering, a,b,c,...,z,aa,bb... Of course, formats may be assigned to any register, not just to (hour) and (min). Sroff keeps the page number in register %, so this request gives Roman numeral page numbers:

```
.af % i
```

15. Indexing, Cross-referencing

If an "index file" is specified when Sroff is invoked, Sroff will copy portions of the input bracketed between .ix and .en directly to that file with no changes except substitution for insertion characters. The index file can be processed later (perhaps sorted, or even Sroffed). Suppose we wish to index the key phrase of this sentence by page number, so that the index file gets a line like this:

```
T-14 - key phrase
```

To do this we used an insertion character followed by % to pick up the page number:

```
Suppose we wish to index the key phrase
.ix
T-^% - key phrase
.en
of this sentence by page number.
```

The index entry came immediately after the phrase to be indexed to assure that the page number would be right.

A table of contents could be produced by index requests imbedded in the input (again, ^ is the insertion character):

```
.ix
^P. Indexing and cross-referencing  <tab>  ^%
.en
```

Cross-references may be remembered similarly by assigning the current page number, equation number, or whatever to a register and calling it out later on. Here (XR1) holds the cross-reference.

```
This is the word.
.an (XR1) ^%
...
The word was given on page ^(XR1).
```

Though backward cross-references are easy, forward cross references pose real obstacles. One clever Sroffer did the job by Sroffing a whole document with printing turned off (by .np 9999 and .so), gathering cross-references, then doing it again with printing turned on (by .np 0 and .so):

```
.np 9999
.so user/document
.np 0
.pa 1
.so user/document
```

This is all very well, except that depending on the size of the numbers inserted, subtle changes in pagination of the document could happen the second time through.

16. Page Offset, Line Numbering

Page offset (.po) requests come in handy when the machine you use for printing places the output too close to the left side of the paper. A page offset request before everything else will move all the output right. In general, .po should not be used in place of .in to affect indentation—it moves headers, footers and everything else with it.

Line numbers may be produced automatically in the margin left by .po (a page offset of 4 will usually suffice). The request .nl causes the lines on each page to be numbered starting from 1, while .n2 causes continuous numbering from the request onward. Line numbering is shut off by .n0. This section was bracketed by .n2 and .n0 requests.

You can retrieve the latest line number by using the name # after an insertion character, and you can set it by .an #. Here is a scheme for causing one line to be unnumbered without interrupting the number sequence. As before ^ is the insertion character:

```
        .an A ^#    save line number in register A
        .n0
        This line is not to be numbered
        .br
        .n2
        .an # ^A    restore line number to previous value
```

17. Text Registers

If you have to use identical bits of text over and over again in a document, you may assign the stuff once to a register, then get it back whenever you need it. For example, to assign the words " Bell

Telephone Laboratories, Incorporated" to a register named " BTL" write

```
.at (BTL)
Bell Telephone Laboratories, Incorporated
.en (BTL)
```

The register is named just like a number register, and the fragment of text is contained between an .at (assign text) request and an .en (end) request on which the name *must* be repeated.

Having put the stuff in a register, use an insertion character (the same character, set by .ic, that was discussed above under " Numbering") with the name to get the text back:

```
.ic ^
The transistor was announced by ^(BTL) in 1947.
```

Sroff automatically replaces " ^(BTL)" by " Bell Telephone Laboratories, Incorporated."

Text registers come in handy to avoid typing the same sequence of requests over and over in elaborate documents, especially when the sequence is rather tricky to get right. This example, where ^ is again the insertion character, sets up the requests for a numbered section heading, named (ph), in the style of this tutorial guide. (The method of numbering was described above under " Numbering" .)

```
.at (ph)
.sp          leave a space
.ne 2        avoid a widow
.an P +1     calculate the number
.ul 2        underline number and following title
^^P.         insert the number
.en (ph)
```

Notice the trickery with the double insertion character. If there were only one, then the text in register (ph) would end up looking like this and would always put out the same number, namely 1.

```
.sp
.ne 2
.an P +1
.ul 2
1.
```

The trouble happens because the ^P gets replaced by a number as the stuff comes in the first time, while being assigned to register (ph). But as we actually wrote it, the double ^^ will be reduced to a single ^ on the first reading*. Having now done the hard work, all we do to start a new section, such as the present one, and to get the proper number attached, is write

```
^(ph)
Text registers.
```

Text registers holding requests are so useful that Sroff has a way to use them just as if they were requests. When the name has two letters, as in the last example, it may be used this way.

```
.ph
Text registers.
```

In short, by using text registers, you can make up your own requests.

18. Parameters

Text registers used as requests can have parameters. For example, suppose we wish to invent an .ip n request to begin an indented paragraph that works just like the sequence

```
.sp
.in n
.ti +10
```

* An explanation of this special handling of the insertion character appears under "Numbering".

We denote the quantity n by #1 to show that it is to be the first parameter of .ip. The character # is a *parameter character* of our own choosing, set beforehand by .pc:

```
.pc #
.at (ip)
.sp
.in #1
.ti +10
.en (ip)
```

Given this definition, #1 will be replaced by whatever appears in later .ip requests, as in

```
.ip 10
.ip +5
```

19. Use of the Ignore Request

Ignore requests (.ig) can be used to exclude text conditionally, usually depending on the value of a number in a register. This sequence turns into " one" or " two" depending on whether the value of register (reg) is 1 or 2.

```
.ic ^
.ig (^(reg))
.en (1)
one
.ig (0)
.en (2)
two
.en (0)
```

If (reg) is 2, then Sroff sees .ig(2) request and ignores everything down to the matching .en(2) (end) request. " two" gets taken as input, and .en(0), which ends nothing at all, gets bypassed. If (reg) is 1, then Sroff ignores only down to .en(1) on the very next line. " one" gets included, then .ig(0) causes " two" to be bypassed.

20. Merge Patterns

Merge patterns accomplish various special effects by putting fixed information into every line. A pattern is given on an input line all by itself, preceded by a .mg request. For example, a table of contents could be Sroffed with a merge pattern of underscores set this way:

```
.mg
-------------------------------------------
```

so the entries would come out looking as below. (Actually a non-blank " blank" , using .tr, will be needed to keep underscores out of the spaces between words.)

```
Complete Table of Requests  _____________ ^(tbl)
```

The vertical bars to the right of this paragraph were set by the following pattern, except that the spacing of the bar was wider.

```
.mg
|
```

21. Multiple Columns

By the mere insertion of .mc 2 in your input text, output will be produced in double-column format. Although any number up to 10 is allowed, the relatively large size of terminal and line printer characters (10 or 12 characters per inch) makes more than three columns appear ungainly. The columns are by default four spaces apart, and each column is of a width such that the right-most one is right justified with respect to the line length which was in effect when single-column output was being generated.

Note that you can adjust the line-length within a column (via .ll-2 and .in+2 in this case). The column offset request can be used to move the second column further to the left or right than the default, just as page offset can be used for the entire page; however, .co flushes pending columns and so should be used only following a .mc.

22. Conclusion

This tutorial guide only hits the high spots. It will get you off the ground, but to get the most out of Sroff, you will want to study the reference manual proper to learn all its capabilities. This can be found in the next chapter, Chapter 6. Happy Sroffing.

Chapter 6

SROFF REFERENCE MANUAL

PAGE

1. Introduction 6-1
2. Usage 6-1
3. Complete List of Sroff Requests 6-13

Chapter 6

SROFF REFERENCE MANUAL

1. Introduction

Sroff is a program to format documents for printing. **Sroff** is similar in nature to **nroff** and **troff**, but it has fewer capabilities and is significantly cheaper to run. A typical **sroff** run costs about one-tenth the cost of the equivalent **troff** run, and produces medium-to-high quality output suitable for internal publication; **troff** is capable of producing superior quality phototypeset output, sometimes required for outside publication.

2. Usage

2.1 Input

Input to **sroff** should be a file with *text lines* containing the information to be formatted, intermixed with *request lines* that contain instructions about how to format it. Request lines begin with a distinguished *control character.*

The syntax of **sroff** is as follows:

sroff [options] [files]

An argument consisting of a minus (-) is taken to be a file name corresponding to the standard input. The *options*, which must appear before the *files* are:

-o*list*
Print only pages whose page numbers appear in the *list* of numbers and ranges, separated by commas. A range *N-M* means pages *N* through *M*; an initial *-N* means from the beginning to page *N*; and a final *N-* means from *N* to the end.

-s*N*

Stop every *N* pages. **Sroff** will halt after every *N* pages (default=N=1) to allow paper loading or changing, and will resume upon receipt of a line-feed or newline.

-m*name*

Prepend to the input file the macro file /usr/lib/smac/m*name*.

2.2 Output

Printed output lines can either be *filled* as nearly as possible with words or can be copied one-for-one from input text. Right-margin justification can be done on filled text. Computation of page numbers is automatic; section numbers, equation numbers, etc., can also be computed. Indentation, centering, line length, line spacing, page layout, titling, hyphenation at line breaks, footnotes, multiple columns, and collecting of an index are all controllable.

2.3 Request Lines

Formatting requests to **sroff** are identified by three characters at the beginning of a line. The first character must be a control character, which is normally the period or dot (.). The short table of requests shown in Figure 6-1 is a good starting set and is adequate for routine formatting.

As the table indicates, some requests cause line *breaks,* that is, termination of the current output line even if it is not filled. Some requests set values and formatting modes; values and modes in effect initially are indicated under " default" .

REQUEST	BREAK	DEFAULT	MEANING
.bp	yes		begin page
.br	yes		break
.ce	yes		center next text line, do not fill it
.ds	yes	no	double space
.fi	yes	yes	fill output lines
.fo t	no	t=""	foot titles are t
.he t	no	t=""	head titles are t
.in n	no	n=0	indent left margin n spaces
.ju	yes	yes	justify right margins of filled lines
.ll n	no	n=60	line length is n, including indent
.nf	yes	no	nofill, break on each input line
.nj	yes	no	no right margin justification
.po n	no	n=0	page offset is n; i.e., move all output n spaces right
.sp n	yes		insert n extra spacing lines
.ss	yes	yes	single space
.ti n	yes		temporary indent, for next line only
.ul	no		underline next input text line

Figure 6-1. A Short Table of Sroff Requests

All the **sroff** requests are given in the section *Complete List of* **sroff** Requests at the end of this chapter.

2.4 Titles

Running titles usually appear at the top and bottom of every page. A title can be set or reset by the request:

.he 'part1'part2'part3'

Part1 is left justified, part2 centered, and part3 right justified with respect to the margins current when the title was set.

There can be up to ten head and ten foot titles for even- and odd-numbered pages. If there are to be several headings the request is:

.he n 'part1'part2'part3'

where n is a number in the range 1 through 10. " .he" could be any one of **.ef, .eh, .he, .fo, .of, .oh**. Any % sign in a title is replaced by the current page number when the title is printed. Any nonblank character may serve as a quote.

The head titles appear in the margin left by **.m2** in the order 1,2,. . ., 10. The foot titles appear in the margin left by **.m3** in the order 10,9,. . ., 1. If the margin is n lines, n<10, titles n+1 through 10 do not appear.

2.5 Blanks

Lines beginning with blank cause a break. Trailing blanks are stripped from input lines. If no characters remain (i.e., the line was all blanks), **.sp 1** is assumed; otherwise, in the output, one blank is appended, or two after '.', '!', '?', or ':'.

2.6 Tabulation

Tab characters in the input specify positions of separate fields on the output line. The fields can be left justified, centered, or right justified. The tab characters are replaced by one or more blanks; these blanks are immune to padding in justified output. The first field is always left justified to the first character position; the positioning of other fields is given by column numbers separated by spaces. Each column number may be followed by L, C, or R to denote left, center, or right tab stops. A column number may be preceded by + or - to cause the tab position to be computed relative to the preceding position. If no numbers are given, the next line after the **.ta** line is taken to contain L's, C's, and R's in the positions of the tab stops. Initially, L-stops are set in every position and the ASCII TAB (octal 11) is the tab character. To use another character (that can be seen in the input file) for the tab character, the request **.tc** can be used.

2.7 Diversions

Certain requests, namely **.at** (assign text), **.fn (footnotes), .ig** (ignore), and **.ix** (index), cause diversions from the main text. Each diversion has an associated label which may be empty or may have the form of a register name described below in Part 2.13, *Register Insertion.* Nested diversions require non-empty end labels. A diversion is ended by a **.en** (end) request having the same label.

2.8 Footnotes

Lines between **.fn** and **.en** are formatted normally, then are held for the bottom of the page. Line formatting is handled completely independently in the footnote text. In particular, separate settings are maintained in the footnote text and in the text proper for **.ce, .li .fi, .in, .ju, .ll, .ls, .mg, .ta, .ti,** and their synonyms and antonyms. The *footnote separator* (normally an empty line) between text body and footnotes is set like a title by **.fs.**

2.9 Translation

Every output character is translated, normally into itself. The request **.tr** cdcd... specifies that each of the characters c is to be translated into the corresponding d. The first c must not be a space. ASCII control characters cannot enter into translation.

2.10 Merging

Fixed information to be put into all output lines may be specified by *merge patterns.* Blank positions of every nonempty output line are replaced from merge pattern 1, then positions still blank are replaced from merge pattern 2, and so on. Merge pattern n is specified by a special input line following a **.mg** n request. The pattern is permanently positioned according to the indentation and page offset (**.po**) current when the pattern was set. A merge pattern containing ASCII control characters will probably not give the intended results.

2.11 Hyphenation

The *hyphenation mode,* set by **.hy** n and **.hp** n, controls attempts to split words at the end of filled lines. Only capitalized words are affected by **.hp**. Hyphenation breaks may be inserted within words:

if n=0, nowhere
if n≥1, at - signs or at hyphenation characters
if n≥2, before certain suffixes
if n≥3, between certain pairs of letters.

The *hyphenation character,* set by **.hc**, disappears from the output. A word containing a - sign or a hyphenation character is treated as if the hyphenation mode were min(n,1).

2.12 Numbering Lines

Line numbers (sometimes useful for debugging input errors) can be produced automatically in the space left by **.po** (a page offset of 4 will usually suffice) under control of **.n0**, **.n1**, and **.n2**. Only nonempty lines not in titles or footnotes are numbered.

2.13 Register Insertion

Sroff can calculate and keep numerical values or text fragments in *registers.* Each register has a 1-to-8 character name chosen from a through z, 0 through 9, %, #, and enclosed in parentheses. Uppercase and lowercase are identical in register names. The parentheses may be dropped from 1-character names. In every input line, appearances of an *insertion character* followed by the name of a register are replaced by the contents of that register. The insertion character is set by **.ic**.

An insertion character not followed by a register name disappears and the following character—even when it is an insertion character—is left untouched. Insertion characters are effective everywhere else, including request lines, diversions, and the interior of parenthesized register names.

2.14 Number Registers

The request **.an** R n assigns the number n to register R. The number must be an unsigned decimal integer, possibly preceded by one of +-*/ to cause modification based on the old value. (That is, +1 means previous value plus 1, etc.) Numeric values are not allowed to go negative; negative values are replaced by zero.

Each register containing a number can have a *format* assigned to it to prescribe how the number is to printed, for example to print a 2 as ii. The request **.af** R f assigns format f to register R. Possible formats and numbers printed in each are:

1	0,1,2,3,4,...,26,27,28,...
01	00,01,02,03,04,...,26,27,28,...
i	0,i,ii,iii,iv,...,xxvi,xxvii,xxviii,...
I	0,I,II,III,IV,...,XXVI,XXVII,XXVIII,...
a	0,a,b,c,d,...,z,aa,bb,...
A	0,A,B,C,D,...,Z,AA,BB,...

Formats 1, 01, 001, . . . specify a minimum width for decimal printing. Shorter numbers in these formats will be padded with leading zeroes. All registers have format 1 initially.

The registers listed below contain special values; all others start at 0.

(year), (mon), (day)	—date and time of day of the start
(hour), (min), (sec)	of the present run, coded as numbers
(%)	—current page number
(#)	—current line number

Register (amon) is initialized to the present month, in English, as in the footer of this manual.

Right after **.bp**, **.ep**, **.op**, or **.pa**, register % contains the current page number. Otherwise it contains the number of the page to which the last character of the last preceding line of input text would be assigned, except in pathological cases where certain requests (**.m1**, **.m2**, **.m3**, **.m4**, **.ne**, **.pl**, **.sk**) unpredictably alter pagination.

> ***Note:*** The % in titles is not preceded by a number character, and is evaluated on output rather than upon input.

Register # contains the line number of the last character of the last preceding line of input text. The request **.an** # n (after a break) will cause the next line to be numbered n+1.

2.15 Text Registers (Macros)

The request **.at** R assigns to register R the following lines of input text, up to a matching **.en** R. The final carriage return is dropped. Considered as a macro, the text assigned to a register may contain parameters, each signaled by a *parameter character* followed by a digit. The parameters are used for argument substitution as described below. The parameter character is set by **.pc** and has no special significance except when followed by a digit under **.at**.

If a text register has a 2-character name, say (xx), then it may be used as if it were a request. Later request lines .xx will be replaced by the text of the register plus a final carriage return. Any arguments—strings on the .xx line separated by blanks and counted 1, 2, . . ., 9 from left to right—will be substituted into the replacement text for occurrences of the parameter character followed by a digit 1 through 9. If, in a text register, the parameter character appears without being followed by a digit, the parameter character disappears on output or rather is replaced by the character following it (even if that second character is the parameter character). Blanks between .xx and the first argument are allowed but unnecessary.

2.16 File Insertion

A *source file* can be introduced in the middle of **sroff** input by a **.so** request giving a filename. Source requests may be nested almost indefinitely, even recursively.

2.17 Canned Programs

These **sroff** programs are permanently stored in the file system to be fetched by **.so**:

.so <defaults> resets all parameters to default values.

.so <corrball> sets up a translation (by **.tr**) to run an EBCDIC Selectric terminal (IBM 2741) using a "correspondence" ball.

Macro packages normally specified on the command line by '\fB-M?', where '?' is any text, can be invoked by **.so** <?mac>.

2.18 Indexing

If an *index file* is specified when **sroff** is invoked, then lines between **.ix** and **.en** are diverted to it. Diverted lines may include references filled in from number registers, but otherwise, the lines are unmodified. Requests other than **.en** are treated simply as text within an indexed region. The lines are ignored if no index file is present.

2.19 Debugging and Caveats

Sroff does not produce error comments, except for file accessing errors. Every effort has been made to make nonsensical requests yield reasonable symptoms. To help in editing, the **.pr** request causes requests to be printed in the right margin along with the output, **.ps** prints original line numbers, and **.uf** shows where original lines started.

Some boundary conditions and pitfalls:

- **Sroff** imposes a limit of 180 characters, counting backspaces, upon each line (or pair of lines in filled text), limits text registers to 400 characters each, and limits the footnotes on any one page to 4000 characters.

- Although it accepts all ASCII characters and escape sequences, **sroff** knows the meaning of only a few—form feed, carriage return, newline, horizontal tab and backspace. In particular, forward and reverse half line feeds will work reasonably only if balanced within the line.

- Control characters usually do not work in merge patterns.

- Backspacing across a blank in filled text almost never produces the desired effect.

- No numeric value is ever permitted to go negative. In particular a negative cumulative indent or a negative register value cannot exist.

- **.ul**, **.us**, **.bf** and **.ce** apply only to text lines not to titles.

- Titles, registers, footnotes, tab settings, and merge patterns use memory and cause **sroff** to grow in space and cost. Macro files add further to cost since prototypes of the macros are read and saved even if never used.

2.20 Processing Sequence

Steps 1 through 5 are performed character-by-character.

1. Read input from current source (file, text register, number register, or insertion argument). If exhausted, pop source and try again.

2. If source is text register, recognize parameter flag (see step 7) and switch source to argument.

3. Recognize insertion character, continue getting characters for name, then switch source (further insertions are honored within a parenthesized name).

4. Replace tab character by unpaddable blanks according to **.ta**, except under **.ix** or **.at**

5. Repeat steps 1 through 4 until one line of input is collected.

Steps 6 to 9 are performed input-line-by-input-line.

6. Inside **.at**, **.ig**, and **.ix** diversions, recognize closing **.en**; otherwise diverted line goes straight to destination. Under **.at**, parameter positions as indicated by **.pc** are replaced by parameter flags.

7. Recognize and perform requests.

8. Underline according to **.ul** and **.uf** and replace hyphen characters by hyphenation flags.

9. Repeat steps 1 through 8 until a line of output is collected.

Steps 10 through 14 are performed output-line-by-output-line.

10. Eject a page if there is not room for the line and insert footnotes, margins, and titles as appropriate.

11. Insert line numbers and request summaries as requested by **.n1**, **.n2**, **.ps**, and **.pr**.

12. Insert combined merge patterns 1,2,3,...,10.

13. Translate according to **.tr**.

14. Append the line, preceded by spacing specified by **.ls**, onto the output file, or onto the footnote collection buffer if under **.fn**, or nowhere if under **.np**.

3. Complete List of Sroff Requests

The following listing contains a description of each **sroff** request. Numerical values are denoted by n or +n, titles by t, and single characters by c. Numbers denoted +n may be preceded by one of +-*/, in which case the the previous value is increased, decreased, multiplied, or divided by n (division by zero yields zero). Otherwise, the request simply replaces the value. No numeric value is allowed to be set negative; attempts to do so cause the value to be set to zero.

Missing n fields are taken to be 1, missing t fields to be empty. Missing c fields turn off **.cc**, **.hc**, **.ic**, **.pc**, and **.tc**. End labels are denoted by e and may be empty or may have the form of a register name described in Part 2.13, *Register Insertion.*

Synonyms are indicated by " syn =" .

Parenthesized defaults are octal representations of nonprinting characters.

REQUEST	*BREAK*	*DEFAULT*	*MEANING*
.ab	yes		abort the run (system debugging only)
.af R f	no	f=1	assign format to register R, f=i,I,a,A,1,01,...
.an R +n	no	n=0	assign number to register R, R≠%; if result is negative, replace by zero
.ar	no	yes	arabic page numerals (syn = **.af** % 1)
.at R	no		assign text to register R until **.en** R
.bc c	no	(010)	c will be treated as a backspace
.bf n	no	n=1	next n lines will appear in **boldface**
.bo n	no	n=1	boldface will be simulated by n overstrikes
.bp	yes		begin page
.br	yes		break
.cc c	no	c=.	control character is c
.ce n	yes	n=1	center next n text lines, break on each
.co +n	yes	n=0	second column offset is n
.di e	yes		divert output to register e until **.en** e
.ds	yes	no	double space (syn = **.ls** 2)

REQUEST	*BREAK*	*DEFAULT*	*MEANING*
.ef n t	no	t=""	n*th* even page foot title is t, 1≤n≤10
.eh n t	no	t=""	n*th* even page head title is t, 1≤n≤10
.en e			end all diversions labeled e, break if end of footnote
.ep	yes		begin an even page
.fi	yes	yes	fill output lines
.fl			flush output buffer (as in **nroff**)
.fn e	no		divert text to footnotes until **.en** e
.fo n t	no	t=""	n*th* even/odd foot titles are t, 1≤n≤10
.fs t	no	t=""	footnote separator is t
.hc c	no		hyphenation character is c
.he n t	no	t=""	n*th* even/odd head titles are t, 1≤n≤10
.hp +n	no	n=1	hyphenation mode for capitalized words is n, 0≤n≤3
.hy n	no	n=3	hyphenation mode is n, 0≤n≤3
.ic c	no		insertion character is c
.ig e	no		ignore all input until **.en** e

REQUEST	*BREAK*	*DEFAULT*	*MEANING*
.in +n	no	n=0	indent left margin n spaces
.ix e	no		divert input to index file until **.en** e
.ju	yes	yes	justify right margin of filled lines
.li n	no		literal, take next n lines to be text
.ll +n	no	n=60	line length is n including indent
.ls +n	yes	n=1	line spacing is n
.lv n	no		leave n consecutive blank lines; wait until next page if necessary
.m1 +n	no	n=4	margin above head no. 1 is n lines
.m2 +n	no	n=2	margin below and including heads is n
.m3 +n	no	n=2	margin above and including feet is n
.m4 +n	no	n=4	margin below foot no. 1 is n lines
.m5 +n	no	n=1	margin for footnote separator is n
.m6 +n	no	n=0	margin below footnote separator is n
.mc n	yes	n=1	multi-column mode is n
.na	yes	no	No adjust (syn = **.nj**, for nroff compatibility)

REQUEST	*BREAK*	*DEFAULT*	*MEANING*
.mg n	no	n=1	next line sets merge pattern n, 1≤n≤10
.n0	yes	yes	do not number output lines
.n1	yes	no	number output lines, reset each page
.n2	yes	no	number output lines, no page reset
.ne n	no		need room for n output lines with present spacing, do **.bp** if necessary
.nf	yes	no	nofill, break on each input line
.nj	yes	no	no right margin justification
.np n	no	no	no printing of output for next n pages
.nu	no	yes	no first character underlining
.oc			overstrike character has been restored. The overstrike character acts as a toggle on boldface within a line; its effect is confined to one line.
.of n t	no	t=""	n*th* odd page foot title is t, 1≤n≤10
.oh n t	no	t=""	n*th* odd page head title is t, 1≤n≤10

REQUEST	*BREAK*	*DEFAULT*	*MEANING*
.op	yes		begin an odd page
.pa +n	yes	n=1	begin page with page number n
.pc c	no		parameter character is c
.pl +n	yes	n=66	paper length is n lines
.po +n	no	n=0	page offset is n, i.e. move all output n spaces right
.pr +n	no	n=0	print requests indented n, don't print if n $\leq$ line length
.ps +n	no	n=0	print sequence numbers of input lines indented n (n for **.pr** > n for **.ps**)
.qc c	no	none	c is argument quote character
.ro	no	no	roman page numerals (syn = **.af** % i)
.sk +n	no		skip at next new page to page number n
.so c/f	no		insert sroff source from filename
.sp n	yes		insert n extra spacing lines
.sp -1			overlay next line on previous if it doesn't collide.
.ss	yes	yes	single space (syn = **.ls** 1)
.ta	no	all	tabs set by this line or next

REQUEST	*BREAK*	*DEFAULT*	*MEANING*
.tc c	no	(011)	c will be treated as a tab
.ti +n	yes		temporary indent, for one line only
.tr cd...	no		translate c into d on output
.uc			Underline character has been restored. Underline character acts as a toggle on underline within a line; its effect is confined to one line.
.uf	no	no	underline first character of each input text line
.ul n	no	n=1	underline alphanumerics in next n input text lines
.us n	no	n=1	underscore all printing characters in the next n input text lines
.ws n	no	n=1	suppress leading and trailing widows

Your comments and suggestions are appreciated and will help us to provide the best documentation for your use.

1. How would you rate this document for COMPLETENESS? (Please Circle)

   Excellent Adequate Poor
   4 ----------3 ----------2 ----------1 ----------0

2. Identify any information that you feel should be included or removed.

   _____

   _____

3. How would you rate this document for ACCURACY of information? (Please Circle)

   Excellent Adequate Poor
   4 ----------3 ----------2 ----------1 ----------0

4. Specify page and nature of any error(s) found in this document.

   _____

   _____

5. How would you rate this document for ORGANIZATION of information? (Please Circle)

   Excellent Adequate Poor
   4 ----------3 ----------2 ----------1 ----------0

6. Describe any format or packaging problems you have experienced with this document.

   _____

   _____

7. Do you have any general comments or suggestions regarding this document?

   _____

   _____

8. We would like to know a little about your background as a user of this document:

A. Your job function _____

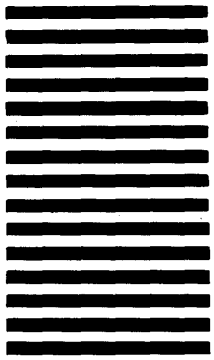
B. Number of years experience with computer hardware: operation _____ , maintenance _____ .

C. Number of years experience with computer software: user _____ , programmer _____ .

Your Name _____ Phone No. _____
Company _____
Address _____
City & State _____ Zip Code _____

Western Electric

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES



BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 1999 GREENSBORO, N.C.

POSTAGE WILL BE PAID BY ADDRESSEE

DOCUMENTATION SERVICES
2400 Reynolda Road
Winston-Salem, N.C. 27106-9989

Do Not Tear—Fold Here and Tape