UNIPLUS+ SYSTEM V
Document Processing Guide

PREFACE

An important feature of the UNIX operating system is to provide a method of document preparation. This guide provides information needed to make use of the system and describes programs that are used to format a document in a user-controlled style. A user should be familiar with the text editing programs (i.e., **ed** or **vi**) of the UNIX operating system before attempting to create a document using the programs described here.

This guide contains six chapters:

- INTRODUCTION
- TROFF TUTORIAL
- NROFF/TROFF FORMATTING PROGRAM
- TABLE FORMATTING PROGRAM
- MATHEMATICS TYPESETTING PROGRAM
- MEMORANDUM MACROS

Chapter 1, INTRODUCTION, gives beginners an overview of the document processing sequence. A beginner should read this chapter before attempting to use the information covered in the other chapters of this guide. An experienced user probably does not need to read this introduction to text processing.

Chapter 2, TROFF TUTORIAL, presents information to enable the user to do simple to complex formatting tasks. When using this document keep in mind that, in most respects, the **troff** formatter is identical to the **nroff** formatter. The main differences are spacing capabilities, font and point size changes, and local motion effects.

Chapter 3, NROFF/TROFF FORMATTING PROGRAM, is a reference guide for using the formatters and making incremental changes to existing packages of commands. It contains tables that list the available requests with defaults and explanations. This chapter is best used in conjunction with the TROFF TUTORIAL or MEMORANDUM MACROS chapters.

Chapter 4, TABLE FORMATTING PROGRAM, describes the **tbl** program usage and the input commands used to generate documents that contain tables. The examples at the end of the chapter are very helpful in setting up tables containing similar features.

Chapter 5, MATHEMATICS TYPESETTING PROGRAM, describes the **eqn** program usage and language for obtaining text with mathematical expressions. The language interfaces directly with the **troff** processor so mathematical expressions can be embedded in the running text of a manuscript and the entire document produced in one process.

Chapter 6, MEMORANDUM MACROS, is a reference manual for the Memorandum Macros package. These macros provide a general purpose package of text formatting macros used with the **nroff** and **troff** formatters. The macros provide users of the UNIX operating system a unified, consistent, and flexible tool for producing many common types of documents. Although the UNIX operating system provides other macro packages for various specialized formats, **mm** is the standard general purpose macro package for most documents such as letters, reports, technical memoranda, released papers, manuals, books, design proposals, and user guides.

CONTENTS

Chapter 1: INTRODUCTION

CONTENTS

Chapter 1
INTRODUCTION

1. Overview

There are three major steps involved in preparing a document on the UNIX operation system:

- Inputting
- Formatting
- Printing

Entering data into a computer system is known as inputting. This input contains the actual text to be printed as well as commands that specify the format of the text. These formatting commands must be converted by a text formatter (**nroff** or **troff**) to information a printer or phototypesetter understands. This process is known as formatting. The **nroff** and **troff** formatters not only format the text, but also send the formatted information to the printer or phototypesetter.

Remember that it takes time to learn to use the document processing tools of the UNIX operating system and don't get discouraged if your document does not turn out exactly as planned. Practice using the text formatters as much as possible even on small projects such as letters and notes to get familiar with the different features and their results. At first it may seem easier to just use a typewriter to perform these tasks, but as your skill increases you will find that using the tools described in this guide will greatly decrease your overall production time.

2. Inputting

The input is composed of the text you want printed as well as the formatter controls which tell the program how you want your text to look on the page. The formatter controls appear in your input file but, once formatted by **nroff** or **troff**, will not appear in your printed output.

Text controls in document preparation can be in the form of:

- Requests
- Macros
- Strings
- Registers

Most documents go through several versions before they are finally finished. Accordingly, you should do whatever possible to make revisions easy. Start each sentence on a new line. Make lines short and break lines at natural place, such as after punctuation like commas and semicolons. These precautions simplify editing since most changes to documents entail rearranging, adding, or deleting sentences and phrases.

Keep file size down to a modest size. Larger files edit more slowly. If any mistakes are made, it is better to ruin a small file rather than the whole document. Split files at natural boundaries such as sections or chapters.

2.1 Requests

A request is an instruction to **nroff** or **troff** that is interspersed in your text to direct the appearance of the output. A text processing request has the following characteristics:

- The request must be entered on a line by itself.
- The request must start either with, generally, a period (.) or, occasionally, an acute accent (´) at the beginning of the line, followed by two lowercase characters.
- The request can be located anywhere in the document but normally affects only the text that follows it.
- The request can have optional arguments located on the same line.

Available requests range from the very simple such as:

```
.sp
```

which will cause a space to appear in your output, to very complex

environments and diversions. Fortunately, all of the requests do not have to be used to prepare a document.

2.2 Macros

A macro is a 1- or 2-character abbreviation (name) that replaces a sequence of formatting requests. This ability to define a group of formatter requests into a single macro is one of the most useful functions of the formatters. The use of macros simplifies the task of formatting a document by allowing the user to define powerful **nroff** and **troff** functions that can be called by a single name and modified easily. As an example, the collection of requests that makes decisions regarding spacing, indentation, and numbering for a paragraph can be replaced with

```
.P
```

on a line by itself. This is done by defining the macro to equal several requests. It is very difficult for a beginner to define their own macros; therefore, a "package" of predefined formatting macros is provided.

2.2.1 Macro Packages

The Memorandum Macro package has a set of macros that have been predefined (collected and named) and make preparing a document easier than using the basic **nroff** and **troff** requests. This macro package enables you, among other things, to create displays, page headers and footers, headings, paragraphs, titles, footnotes, lists, and multicolumn output. It is easier to learn how to use a macro package than to set up a document by using the **nroff** and **troff** requests exclusively.

Formatting macros typically consist of a period and one or two uppercase letters, such as

```
.P
```

that is used to begin a new paragraph or

```
.BL
```

to initialize a bullet list.

2.3 Strings

A string is a 1- or 2-character variable that is embedded in the text or in a macro definition. It is actually a text register that is defined to contain a string of characters that can be printed by simply calling the

string name. Sequences of words or characters that occur repreatedly in a document can be replaced by a string.

The contents of a string are printed by entering *x where x is a single-character name of the string or *(xx where xx is a two-character name of the string. For example, if the name of a company appears several times throughout your document, you could define a string using the **.ds** request as shown:

```
.ds SS Smart Software Corporation
```

Therefore, whenever the company name should appear in your output, the following can be substituted:

```
The \*(SS is located in Arkansas.
```

and will output:

```
The Smart Software Corporation is located in Arkansas.
```

Note that a string must be defined before it can be used in the text. It is a good practice to define strings at the beginning of your document.

2.4 Registers

Text processors provide three different kinds of registers:

- Predefined general number registers.
- Predefined read-only number registers.
- User-defined number registers.

The predefined registers have default values. These registers are maintained by the text formatters. They are used to define part of the overall appearance of your document. A general number register can be read, written, automatically incremented or decremented, and interpolated into the input. Number registers may also be used in numerical expressions, for flags, and for automatic numbers.

For example, in the Memorandum Macros the appearance of a paragraph is controlled by the the registers **Pt** (type), **Pi** (indent), and **Ps** (spacing). To set the type of paragraph to be indented with an indent of 3 and 2 spaces preceding it, the following would be used:


```
.nr Pt 1
.nr Pi 3
.nr Ps 2
```

Number registers tend to be too complicated for a beginning user to take advantage of their usage.

3. Formatting

Once you have created a file of text, you are ready to format it. Text processors prepare your files of text for printout on printers and photo-typesetters.

The **nroff** program formats files for printing on typewriter-like devices (low-speed, letter-quality printers) and line printers as well as output on a terminal. The **nroff** program formats text into a printable paginated document.

The **troff** program formats files for printing on a typesetter. It is designed to drive a typesetter that produces high-quality output on photographic paper. This document was formatted with **troff**.

The basic ideas of formatting programs is that the text to be formatted contains with it "formatting commands" that determine in detail how the text is to look. There may be commands that specify the line length, margins, single- or double-spacing, page numbering and titles to use on each page.

3.1 Using Format Programs

The input form for invoking formatting programs is:

nroff *options filenames*

or

troff *options filenames*

where *options* are arguments to **nroff** and **troff** (see **nroff**(1) and **troff**(1) in the *UniPlus*$^+$ *User's Manual*) and *filenames* are the names of the files containing the document to be formatted.

To produce a document in standard format using the Memorandum Macros, use the following:

nroff −**mm** *filenames*

for output on a terminal and

troff −**mm** *filenames*

for a typesetter.

4. Preprocessors

In the same way that macro packages make it easier to create a document than the detailed **nroff** and **troff** programs, so do the preprocessors make complicated operations simplier to perform.

The **tbl** program converts files containing tables for **nroff** and **troff** output. The **eqn** program converts files containing mathematical equations and expressions for **troff**, and the **neqn** program converts the same for **nroff**.

The tables and equations can be interspersed in your text. Each preprocessor has special names to define the beginning and end of input for it as follows:

PREPROCESSOR	START MACRO	END MACRO
eqn	.EQ	.EN
neqn	.EQ	.EN
tbl	.TS	.TE

The preprocessor converts the lines between the start and end macros into text processing requests to be read by **nroff** or **troff**.

To produce a document using the preprocessors, use the following:

neqn *options filenames* | **nroff** *options*

or

eqn *options filenames* | **troff** *options*

tbl *options filenames* | **nroff** *options*

or

tbl *options filenames* | **troff** *options*

To produce a document using both preprocessors, use the following:

tbl *options filenames* | **neqn** | **nroff** *options*

or

tbl *options filenames* | **eqn** | **troff** *options*

Chapter 2: TROFF TUTORIAL

CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 2

TROFF TUTORIAL

1. Overview

Troff is a text-formatting program for driving a phototypesetter on the UNIX operating system. It specifically formats text for a Wang Laboratories, Inc., C/A/T phototypesetter, but interfaces have been written to adapt **troff** to other devices. High quality text can be produced with **troff** since its capabilities include dynamic font and point-size control, a full Greek alphabet, special characters, mathematical symbols, and horizontal/vertical local motions at any point.

An important rule of using the **troff** formatter is to use it through an intermediary such as a macro package. For one page layouts such as announcements and forms where it is necessary to have complete control over spacing, it is sometimes useful to use **troff** without a macro package. 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 too difficult for most people to use effectively.

For special applications several programs provide an interface to the **troff** formatter for the majority of users.

- 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. (See the "Mathematics Typesetting Program" in the *Document Processing Guide.*)
- The **tbl** program provides an easy to learn language for producing tables of arbitrary complexity. (See the "Table Formatting Program" in the *Document Processing Guide.*)
- The **mm** package contains a range of commands from easy to complicated that are useful for formatting many different styles of documents.

For producing text that may contain mathematics or tables, 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. (See the "Memorandum Macros" in the *Document Processing Guide.*) There are also packages for viewgraphs and other special applications. These packages are easier to use than the **troff** formatter alone once the user gets beyond the most trivial operations. They should be considered first.

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

To use the **troff** formatter, the user must prepare actual text 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, in no-fill mode {2}:

```
Some text.
.ps 12
Some more text.
```

will change the point size of the letters being printed to 12 point (one point is 1/72 of an inch) as below:

Some text.
Some more text.

Occasionally, something special occurs in the middle of a line, such as an exponent. The formula for the area of a circle is input as follows:

```
Area = \(*p\fIr\fR\|\s7\u2\d\s0
```

and will produce:

Area = πr^2

The backslash (\) is used to introduce **troff** commands and special characters within a line of text.

A note concerning the formatting of this document: throughout the text of this document, UNIX-specific words will appear in **bold** and *italics* will be used to designate variable information and emphasis. Special-meaning words will be in quotes. Command lines will be indented with information to be typed as it appears in Roman. Numbers enclosed in braces ({}) refer to section numbers within this document. Table numbers correspond to the section in which they are primarily referred to. If there are two or more tables in one section, an alphabetic level is used.

2. Text Filling and Adjusting

The **troff** formatter collects the words from the text input lines (ignoring the layout of the lines) and assembles them to fill up to the current output line length. When a word does not fit on that line, there is a break and the text is then begun on the next line. This is "filled" text and is a **troff** default feature. In "no-fill" mode **troff** outputs the exact layout of the input lines. The no-fill mode can be set with the **.nf** request and returned back to fill mode with the **.fi** request.

Troff will fill the remaining space left by the break by increasing the space between the words on that line. This is "adjusted" text and and is a **troff** default feature. To prevent the adjustment of the line spaces, the **.na** (no adjust) request is used and to return back to adjusted text, the **.ad** request is used.

This document reflects the **troff** default features of filled and adjusted text.

3. Point Sizes and Line Spacing

The **.ps** request and the \s sequence set the point size of the characters. 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 – 6, 7, 8, 9, 10, 11, 12, 14, 16, 18, 20, 22, 24, 28, and 36. The default of **troff** processor is point size 10. This document has been typeset in 9 point.

If the number following the **.ps** request is not a legal value, the point size is rounded up to the next valid value with a maximum of 36. If no number follows the **.ps** request, point size reverts to the previous value.

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. The \s0 sequence causes the size to revert to its previous value. The "\s1011 apples" sequence is understood correctly as "point size 10, followed by the text, 11 apples".

Relative size changes, ± the current point size, are also legal and useful:

```
\s-2AUNT\s+2 and \s-2UNCLE\s+2
```

produces

AUNT and UNCLE

temporarily decreasing the size by two points, then restoring it. Relative size changes have the advantage that the size difference is independent of the starting size of the document. Therefore, the point size of this entire document can be changed but the above example will still be 2 point sizes smaller. 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. Two points is a good space difference between text lines. For example, this document was typeset with the following combination:

```
.ps 10
.vs 12p
```

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** formatter uses a point size of 10 and a vertical spacing of 12 points. When **.vs** is used without arguments, vertical spacing reverts to the previous value.

The **.sp** request is used to obtain blank vertical space. Used without arguments, **.sp** causes one blank line to be output (at whatever value **.vs** is set). Most **troff** output devices can interpret fractions. Arguments to **.sp** can be scaled in the following ways:

TABLE 3. Argument Scales to **.sp**

SCALE	DESCRIPTION	USAGE
omitted	vertical space of current .vs	.sp 2
v	vertical space of current .vs	.sp 3v
i	inches	.sp .5i
p	points	.sp 12p

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 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).

4. Fonts and Special Characters

The **troff** processor and the typesetter allow four different fonts at one time. Typically, these four fonts are permanently mounted on the physical typesetter in positions 1, 2, 3, and 4, respectively:

R Times Roman,
I Times Italic,
B Times Bold, and
S Special Mathematics.

The Special Mathematics font contains the Greek alphabet, mathematical symbols, and miscellaneous symbols. Characters on the special font are automatically handled by **troff** 4-character input names (see Table 4); therefore, it is not necessary to request the special font.

The default font in **troff** is Roman. To change the current font, the **.ft** request is used. The argument to the **.ft** request can be either the

corresponding letter or number of the font wanted. For example, the input in no-fill mode:

```
.ft B
This command changes the font from Roman to bold.
.ft I
This command changes the font from bold to italics.
.ft R
This command changes the font from italics to Roman.
.ft P
This command returns to the previous font (in this case, italics).
.ft
This command also returns to the previous font (in this case, Roman).
```

will output:

This command changes the font from Roman to bold.
This command changes the font from bold to italics.
This command changes the font from italics to Roman.
This command returns to the previous font (in this case, italics).
This command also returns to the previous font (in this case, Roman).

The above fonts are shown in Fig. 4 at the end of this document. Depending on the phototypesetter being used, there are other fonts available besides the standard set of Times, although only four can be used at any given time. The **.fp** request can be used to rename the fonts that are actually mounted on the typesetter. This would be helpful, for instance, if an extra Roman font is mounted in place of the bold font. For example:

```
.fp 3 H
```

renames the font mounted on position 3 to **H** instead of the default **B**. Appropriate **.fp** requests should appear at the beginning of a document if standard fonts are not used.

Using font numbers instead of letters makes it possible for a document to be relatively independent of the actual fonts used to print it. Therefore, the **.ft 3** request is interpreted as "change to the font mounted in position 3". Numbers and letters are interchangeable and both can be used throughout a document.

Fonts can be changed within a line or word with the \f escape sequence followed by R, I, B or 1, 2, 3. For example, the input:

```
A \fBbold\fR word stands out; a word in \fIitalics\fR shows emphasis.
```

will output:

A **bold** word stands out; a word in *italics* shows emphasis.

The **.bd** request will artificially embolden a font by overstriking letters with a slight offset. (See Section 4.1, "Fonts," in the *Nroff/Troff Formatting Program.*)

In **troff** 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.

The characters on the Special Mathematics font can be accessed by their 4-character input name. These names consist of characters preceded by the \(sequence. These names may be inserted anywhere in the text. In particular, Greek letters are all of the form \(*x, where *x* is an uppercase or lowercase Roman font letter reminiscent of the Greek. A list of these special names is given in Table 4 at the end of this document.

In **troff** 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. Since a backslash (\) is an escape character with a special meaning in **troff**, to get an actual backslash to print a \e or \\ entry must be input.

4.1 Indents and Line Lengths

The default line length in **troff** is 6.5 inches. To reset the line length the **.ll** request is used. Line length can be specified in several scales; inches are probably the most suitable. Usable page width on the phototypesetter is about 7.54 inches.

Page offset is the space from the left edge of the paper to the beginning of the line of type (e.g., the left margin). The default margin is slightly less than 1 inch. The **.po** request is used to set the page offset.

Basically, to get centered text the line length plus twice the page offset (right and left margins) should equal the paper width. For instance the line length and page offset of this 6-inch-wide page is:

```
.ll 4.5i
.po .75i
```

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. To obtain an offset block of text, the **.in** request can be used to move the left margin to the right and a negative argument to the **.ll** request can be used to move the right margin to the left. As an example:

```
.in 1i
.ll -1i
This block of text will be indented by one inch
both on the left margin and on the right margin.
This is a useful method for setting off quotes
or figures from the rest of the text.
The initial .in request does not need to
be incremented.
.ll +1i
.in -1i
```

will output:

> This block of text will be indented by one inch both on the left margin and on the right margin. This is a useful method for setting off quotes or figures from the rest of the text. The initial **.in** request does not need to be incremented.

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 to the right
- **.ll −1i** makes lines 1 inch shorter to the left
- **.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. A **.ti** request can be used to begin a paragraph. If no units are specified the line is indented by the default (three ems). 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.

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 point size *n* is *n* points. Although inches are usually clearer than ems to people who are not used to setting type, 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. For instance, an indent of 3 ems in this document (point size 9) is smaller than an indent of 3 ems in the same document set in the default point size of 10. To make the measurement clearer to the user, the ems can be specified as scale factors, as in **.ti 2.5m**.

5. Tabs

A tab (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 (no-fill mode) text. Tab stops are set by default every half inch from the current indent but can be changed by the **.ta** request. Tabs are specified with numeric measurement as below but can also be specified with the local motion sequence of \w {5.2}. 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) by default. The characters **R** and **C** placed after the tab setting will produce right-justified and centered columns, respectively. In the following examples the ⌑ symbol represents a tab character. The input:

```
.ta 1iC 2iR
first① first① first
second① second① second
third① third① third
```

will output:

```
first          first          first
second        second        second
third          third          third
```

Note that the initial column is not a tabbed column and therefore can not be specified.

Numeric columns can be lined up and evenly spaced by preceding each number with enough blanks to make it line up when printed (in **troff** all digits have the same width). To make a nonprinting *unpaddable* blank space, the escape sequence **\0** is used {5.2} The input:

```
.nf
.ta 1i 2i
\0\01① \0\02① \0\03
\040① \050① \060
700① 800① 900
.fi
```

will output:

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

This will only work with numbers. Trying to use this method with words will be time consuming and most likely ineffective.

The table formatting program (**tbl**) is the best and fastest way to format tables and would make the formatting of the most complicated table easier than the above-mentioned method. The **troff** processor provides a general mechanism called "fields" for setting up complicated columns. This is used by the **tbl** program.

Tab space can be filled up with some character other than a blank. This can be done with the **.tc** request which sets a tab replacement character.

An underline (**\(ru**) is the most common, but any character (such as a period) can replace the blank character. The **\0** sequence can be used to make a space between the lines drawn and the text. For example, the input:

```
.nf
.ta 2i 3i
.tc \(ru
Name\0① \0Age\0①
Address\0① ①
.fi
```

will output:

Name _____ Age _____
Address _____ _____

To reset the tab replacement character to a blank, use the **.tc** request with no argument. Another way lines can be drawn is with the **\l** escape sequence as described in section 6.4.

6. 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. A good rule to follow is to make uniform and logical sequences. For instance, if a string is to be made a point size smaller, in italics and with a subscript, the input could be:

```
\s-1\fIstring\d1\u\fR\s+1
```

as opposed to

```
\fI\s-1string\d1\fR\u\s+1
```

Both versions are correct but the first example reads better and makes it easier for a user to find errors.

6.1 Vertical Motions

The **eqn** program is more useful than **troff** for formatting mathematical equations. If the **eqn** program is not used, subscripts and superscripts are most easily done with the half-line local motions **\u** (up) and **\d** (down) sequences. To go back up the page half the current point size,

insert a \u at the desired place; to go down half the current point size, insert a \d. The \u and \d should always be used in pairs because all of the text following the escape sequence will continue to be either up or down half a point size. So, for every time something is moved up half a space it has to be moved back down again. 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 exact amount wanted. 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*. The ' symbols are used as parameters. A minus sign causes upward motion, while no sign or a plus sign means down the page. For example, the input:

```
\v'1'\s18D\s0\v'-1'own
```

will output:

D$^{\text{own}}$

Thus **\v'1'** causes a downward vertical motion of one line space and **\v'-1'** causes an upward vertical motion of one line space.

There are several ways to specify the amount of motion:

```
\v'0.1i'      (inches)
\v'3p'        (points)
\v'-0.5m'     (ems)
```

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.

As with the \u and \d escape sequences, the \v should always balance vertical motions in a line with the same amount in the opposite direction.

6.2 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. A minus sign causes a backward (to the left) motion, while no sign or a plus sign causes a forward (to the right) motion. As an example,

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

causes a backwards 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 specific character string. The construction

```
\w'thing'
```

is a space 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 a *word*:

```
\h'\w'word'u
```

is used. Since the default scale factor for all horizontal dimensions is **m** (ems), **u** must be used to denote machine units. If the **u** is not specified, the motion produced will be computed in ems and therefore 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 "boldface" in bold by overstriking with a slight offset. The following example prints "boldface", moves left by the width of "boldface", moves right one unit, and prints "boldface" again. The input:

```
boldface\h'-\w'boldface'u'\h'1u'boldface
```

will output:

boldface

Section 11 describes a way of avoiding typing so much input for each command name.

As another example of the \w sequence, to set a tab the width of the longest word in a list plus two spaces:

```
.ta \w'longest\ \ 'u
long① 1
longer① 2
longest① 3
```

will produce:

```
long    1
longer  2
longest 3
```

The following are special-purpose **troff** escape sequences for local motion that are most useful when exact spacing is needed:

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

6.3 Overstrikes

The **\o** sequence causes characters to be overstruck. Up to nine characters can be listed within parameters and all will be overstruck centered on the widest character. For example,

```
\o'n~'                  produces  ñ
\o'>|'                  produces  ≯
r\o'e\(aa'sum\o'e\(aa'   produces  résumé
\o'\s+1\(ci\s-1|\-'     produces  ⊕
```

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**, all the characters are centered on the widest character, and there can be no horizontal or vertical motions. The **\z** may be the only way to get what is needed. For example,


```
\z\(<-\u\(->\d      produces  ⇄
\z\(ua\d\(->\u      produces  ↥
```

A more ornate overstrike is given by the bracketing function **\b**, which piles up characters vertically, centered on the current base line. Thus big brackets are obtained by constructing them with piled-up smaller pieces using their special 4-character names. For example, the input:

```
\b'\(lt\(lk\(lb' \b'123' \b'\(rt\(rk\(rb'
```

will output:

```
⎧1⎫
⎨2⎬
⎩3⎭
```

6.4 Drawing Lines

A convenient facility for drawing horizontal (**\l**) and vertical (**\L**) lines of arbitrary length with arbitrary characters is provided by the **troff** formatter. A 1-inch long line is printed with a **\l'1i'** sequence. If the _ is not appropriate, the length specification can be followed by a different character. 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 **tbl** program describes other ways of providing horizontal and vertical lines (See *Table Formatting Program*).

7. Strings

If a document 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.

Strings are defined with the **.ds** request. A reference to a string in the text is replaced by whatever the string was defined as. 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. As another example:

.ds ux \s-1UNIX\s+1

can be referenced by the string name *(ux to produce the the string UNIX.

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.

8. Introduction to Macros

Some of the information in this section and the sections following deals with **troff** capabilities that will probably be too involved for the beginning or intermediate user. These sections will, however, give a better understanding of the macro packages available and help in adapting and implementing them.

In its simplest form, a macro is a shorthand notation similar 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. Names are restricted to one or two characters. The last line (..) marks the end of the definition. In between are the requests which are 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. It is a good practice to have all macro definitions at the beginning of the file. If the same macros are to be used in several files, they can be put in a separate file. The name of a file containing the macros can precede the file name of the document being processed when invoking the **troff** formatter at the command level.

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 name
.sp 2p          \"space 2 points
.ti +3m         \"temporary indent of 3 ems
.ft R           \"set font to Roman
..              \"end of macro definition
```

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).

An example of macros that start and end a block of offset, unfilled text is

```
.de OS          \"offset start macro name
.sp             \"one vertical space
.nf             \"no-fill mode
.in +0.5i       \"indent one-half inch to the right
..              \"end macro definition
.de OE          \"offset end macro name
.sp             \"one vertical space
.fi             \"fill mode
.in -0.5i       \"indent one-half inch to the left
..              \"end macro definition
```

The **.OS** and **.OE** macros can be used before and after text to provide an indented block of unfilled text. 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.

9. 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

specifications must be provided:

- what to do before and after the title line,
- when to print the title, and
- what the actual title is.

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

```
.de NP                          \"define new page
'bp                             \"begin page
'sp 0.5i                        \"space one-half inch
.tl 'left top'center top'right top'  \"prints title
'sp 0.3i                        \"space 3/10 inch
..                              \"end definition
```

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 three-part 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 could occur when a new page is started. The effect would be to print the left-over part of the interrupted line at the top of the page, followed by a new output line of the input line that followed the break. 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 but should be stored until the new page macro is executed.

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

.bp	begin page	**.nf**	no-fill mode
.br	break	**.sp**	space
.ce	center	**.in**	indent
.fi	fill mode	**.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
'bp
'sp 0.5i
.tl 'left top'center top'right top'
'sp 0.3i
..
.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.

When changing fonts or point sizes, beware of 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 point size to 10
.lt 6i                             \"set title length to 6 inches
.tl 'left top'center top'right top'
.ps                                \"revert to previous point size
.ft P                              \"revert 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 section 13.

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:

```
.de NP
'sp .5i
.tl ''- % -''    \"center page number inside hyphens
'bp
   --- header stuff
..
```

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**.

10. 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 \n*x* (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 a number from one to four).

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          \"define paragraph macro
.ps \\n(PS      \"reset size
.vs \\n(VSp     \"reset spacing
.ft R           \"set font to Roman
.sp 0.5v        \"space half a line
.ti +3m         \"temporary indent of 3 ems
.."             \"end macros definition
```

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) 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 3½ 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 **.ll 7i/2u** must be used. 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.

11. Macros With Arguments

Arguments are variable information that will be incorporated into a macro. The arguments are on the same line following a macro call. They enable flexibility from one use of a macro to the the next. Two things are needed to be able to define macros with arguments:

- When the macro is defined, it must be indicated that some parts will be provided as arguments when the macro is called.
- When the macro is called, the actual arguments to be plugged into the definition must be provided. Arguments not provided when

the macro is called are treated as empty.

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 text.

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 rearranged and printed in the normal size:

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

This macro defines **.SM** as taking the third argument and placing it at the beginning, the first argument is made two points smaller and put in the center, and the second argument is restored to the previous point size and placed last. The macro call as defined above

```
.SM ABLE ),
```

would output:

ABLE),

The macro call

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

produces output:

(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." 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 section heading number
.ps \\n(PS-1            \"decrease PS number by 1
\\n(SH. \\$1            \"number and 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 calls the three strings named LT, CT, and RT {7}. 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 definitions for any of the strings.

12. 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     \"if SH equals 1, then space 2 inches
```

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    \"if SH is not greater than 1, do .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 .ta 0.75i 1.5i 2.25i
.if n .ta 1i 2i 3i
```

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.

13. 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 potential **troff** 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.

14. 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 new 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                         \"revert to no-fill mode
.XX                         \"print 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.

15. Macro 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. However, there are macro packages available that have done all the calculating already and can be adapted to individual needs with much less work than starting from scratch. Such common formatting needs such 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 could 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.

15.1 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 (space one inch from top of page) and footer (begin new page when text is one inch from bottom of page) 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. Therefore, it is a good practice to put macro definitions and traps at the beginning of a file unless they are specifically needed elsewhere in the document. 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 of a header and footer (a page number at the bottom of the first page and at the top of the remaining pages) follows:

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

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

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

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

15.2 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              \"define paragraph
.br                 \"break
.ft R               \"force Roman font,
.ps 10              \"point size 10,
.vs 12p             \"vertical spacing 12 points,
.in 0               \"and no indent
.sp 0.4             \"prespace
.ne 1+\\n(.Vu       \"need more than 1 line of space
.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              \"define section
   ---              \"force font, point size, etc.
.sp 0.4             \"prespace
.ne 2.4+\\n(.Vu     \"need 2.4+ lines
.fi
\\n+S.
..
.nr S 0 1           \"initial S (section)
```

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 indented paragraph in which a label protrudes left into the indent space.

```
.de lp              \"define labeled paragraph
.pg
.in 0.5i            \"paragraph indent of one-half inch
.ta 0.2i 0.5i       \"tab set for label at .2 inch, paragraph at .5 inch
.ti 0
\t\\$1\t\c          \"label is first argument with tab before and after
.."
```

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

at 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.

15.3 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              \"define header
  ---
.nr cl 0 1          \"initial column count
.mk                  \"mark top of text
..
.de fo \"define footer
.ie \\n+(cl<2 \{\
.po +3.4i           \"next column; 3.1+0.3
.rt                 \"return 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.

15.4 Footnote Processing

The footnote mechanism is used by imbedding 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                  \"define 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                  \"define 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                  \"define footnote overflow process
.if \\nx .di fy         \"divert overflow
..
.de fn                  \"define 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                  \"define 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                               \"define separator
\l'1i'                               \"1 inch rule
.br
..
.de fz                               \"define 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.

15.5 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              \"define 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.

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 ς τ υ ϕ χ ψ ω
Γ Δ Θ Λ Ξ Π Σ Υ Φ Ψ Ω
√ ⩾ ⩽ ≡ ∼ ≃ ≠ → ← ↑ ↓ × ÷ ± ∪ ∩ ⊂ ⊃ ⊆ ⊇
∞ ∂ § ∇ ¬ ∫ ∝ ∅ ∈ ‡ ☛ ☚ 🔔 | ○ ⌈ ⌉ ⌋ ⌊ ⟨ ⟩ | ⌊ ⌋ ⌈ ⌉ |

Figure 4. Font Style Examples

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 by Wang Laboratories, Inc., of Hudson, New Hampshire. The Times Roman, Italic, and Bold are among the many standard fonts available.

TABLE 4. Naming Conventions for Non-ASCII Characters

Non-ASCII characters and *minus* on the standard fonts.					
CHAR	INPUT NAME	CHARACTER NAME	CHAR	INPUT NAME	CHARACTER NAME
’	’	close quote	fi	\(fi	fi
‘	‘	open quote	fl	\(fl	fl
—	\(em	¼ Em dash	ff	\(ff	ff
-	-	hyphen or	ffi	\(Fi	ffi
-	\(hy	hyphen	ffl	\(Fl	ffl
−	\-	current font minus	°	\(de	degree
●	\(bu	bullet	†	\(dg	dagger
□	\(sq	square	′	\(fm	foot mark
_	\(ru	rule	¢	\(ct	cent sign
¼	\(14	one-fourth	®	\(rg	registered
½	\(12	one-half	©	\(co	copyright
¾	\(34	three-fourths			

TABLE 4. Naming Conventions for Non-ASCII Characters (continued)

Non-ASCII Greek characters on the special font.					
CHAR	INPUT NAME	CHARACTER NAME	CHAR	INPUT NAME	CHARACTER NAME
A	\(*A	Alpha†	α	\(*a	alpha
B	\(*B	Beta†	β	\(*b	beta
Γ	\(*G	Gamma	γ	\(*g	gamma
Δ	\(*D	Delta	δ	\(*d	delta
E	\(*E	Epsilon†	ϵ	\(*e	epsilon
Z	\(*Z	Zeta†	ζ	\(*z	zeta
H	\(*Y	Eta†	η	\(*y	eta
Θ	\(*H	Theta	θ	\(*h	theta
I	\(*I	Iota†	ι	\(*i	iota
K	\(*K	Kappa†	κ	\(*k	kappa
Λ	\(*L	Lambda	λ	\(*l	lambda
M	\(*M	Mu†	μ	\(*m	mu
N	\(*N	Nu†	ν	\(*n	nu
Ξ	\(*C	Xi	ξ	\(*c	xi
O	\(*O	Omicron†	o	\(*o	omicron
Π	\(*P	Pi	π	\(*p	pi
P	\(*R	Rho†	ρ	\(*r	rho
Σ	\(*S	Sigma	σ	\(*s	sigma
			ς	\(ts	terminal sigma
T	\(*T	Tau†	τ	\(*t	tau
Υ	\(*U	Upsilon	υ	\(*u	upsilon
Φ	\(*F	Phi	ϕ	\(*f	phi
X	\(*X	Chi†	χ	\(*x	chi
Ψ	\(*Q	Psi	ψ	\(*q	psi
Ω	\(*W	Omega	ω	\(*w	omega

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

TABLE 4. Naming Conventions for Non-ASCII Characters (continued)

Non-ASCII characters and ´, `, _, +, −, =, and * on the special font.					
CHAR	INPUT NAME	CHARACTER NAME	CHAR	INPUT NAME	CHARACTER NAME
+	\(pl	math plus	*	\(**	math star
−	\(mi	math minus	|	\(or	or
±	\(+−	plus-minus	/	\(sl	slash
×	\(mu	multiply	§	\(sc	section
÷	\(di	divide	´	\(aa	acute accent
=	\(eq	math equals	`	\(ga	grave accent
≥	\(>=	greater than or equal	_	\(ul	underrule
≤	\(<=	less than or equal	→	\(−>	right arrow
≡	\(==	identically equal	←	\(<−	left arrow
≅	\(~=	approximately equal	↑	\(ua	up arrow
~	\(ap	approximates	↓	\(da	down arrow
≠	\(!=	not equal	‡	\(dd	double dagger
√	\(sr	square root	Ⓑ	\(bs	Bell System logo
	\(rn	root en extender	☚	\(lh	left hand
∪	\(cu	cup (union)	☛	\(rh	right hand
∩	\(ca	cap (intersection)	|	\(br	box vertical rule
⊂	\(sb	subset of	○	\(ci	circle
⊃	\(sp	superset of	┃	\(bv	bold vertical
⊆	\(ib	improper subset	⌈	\(lc	left ceiling (bracket)
⊇	\(ip	improper superset	⌉	\(rc	right ceiling
∈	\(mo	member of	⌊	\(lf	left floor
∅	\(es	empty set	⌋	\(rf	right floor
∞	\(if	infinity	{	\(lt	left top (brace)
∂	\(pd	partial derivative	}	\(rt	right top
∇	\(gr	gradient	⎩	\(lb	left bottom
∫	\(is	integral sign	⎭	\(rb	right bottom
∝	\(pt	proportional to	⎨	\(lk	left center
¬	\(no	not	⎬	\(rk	right center

Chapter 3: NROFF/TROFF FORMATTING PROGRAM

CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 3

NROFF/TROFF FORMATTING PROGRAM

1. Introduction

This document is not geared toward the beginner but toward a user who is already familiar with using macro packages and is interested in altering or writing macros. It is also a useful reference for **nroff** and **troff** commands that are not available in existing macro packages.

Text processors, **nroff** and **troff**, under the UNIX operating system format text for typewriter-like terminals and for a phototypesetter, respectively. Both **nroff** and **troff** processors accept lines of text interspersed with lines of format control information. They format the text into a printable, paginated document having a user-designed style. The **nroff** and **troff** formatters offer unusual freedom in document styling including:

- Versatile paragraph and section control
- Flexible-style headers and footers
- Generation of footnotes
- Automatic sequence numbering for paragraphs and sections
- Multiple column output
- Font and point-size control (**troff** only)
- Arbitrary horizontal and vertical local motions at any point
- Overstriking, bracket construction, and line drawing functions.

Since **nroff** and **troff** formatters are reasonably compatible, it is usually possible to prepare input acceptable to both. Conditional input is provided that enables the user to embed input expressly destined for either program [17]. For example,

```
.if n .sp      \"if nroff, then go one space
.if t .sp .5   \"if troff, then go one-half space
```

The major dissimilarity between the two formatters is regarding spacing. **Nroff** does not have fractional-space capabilities. A **troff** vertical-space request such as **.sp .5** will be ignored or **.sp 1.3** will be treated as one space by **nroff**. Keep in mind that **nroff** output devices use constant-width characters, whereas in **troff**, character widths vary. This is

important when determining distances for setting tabs. Local-motion escape characters also have different effects in **nroff** and **troff** {12.1}.

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 **troff** processor is a text-formatting program for driving a phototypesetter on the UNIX operating system. **Troff** specifically formats text for a Wang Laboratories, Inc., C/A/T phototypesetter, but there have been interfaces written to adapt **troff** to other devices. It is capable of producing high quality text. The 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.

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.

A note concerning the formatting of this chapter: throughout the text, UNIX-specific words will appear in **bold** and *italics* will be used to designate variable information and emphasis. Special-meaning words will be in quotes. Command lines will be indented with information to be typed as it appears in Roman. Numbers enclosed in braces ({}) refer to section numbers within this chapter. Tables numbers correspond to the section in which they are primarily referred to. If there are two or more tables in one section, an alphabetic level is used. The request tables appear within the section they are mentioned. Tables that are useful formatting tools are placed at the end of this chapter for easy reference.

2. Usage

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

nroff *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 and Troff Options

OPTION	EFFECT
−**o***list*	Prints only pages whose page numbers appear in *list*, which can consist of comma-separated numbers and/or number ranges. • A list of comma-separated numbers such as N,M means pages N and M. • 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 and cause the bell control character to be output to the terminal. 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 **troff** formatter will stop the phototypesetter every *N* pages, produce a trailer to allow changing cassettes, and resume after the phototypesetter START button is pressed.
−**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 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* should be used if it exists. If not, the **nroff/troff** formatter will try the equivalent **−m** *name* option instead. This option should be used instead of **−m** because it makes the **nroff/troff** formatters execute significantly faster.

−r*xN* Set register *x* (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/troff** formatter. This option has no effect if no **.co** request is used in the **nroff/troff** formatter input. Otherwise, the compacted output is produced in files *d.name* and *t.name*.

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, and **450** for the DASI 450.
−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 Only Options

OPTION	EFFECT
−t	Direct output to the standard output instead of the phototypesetter.
−f	Refrain from feeding paper and stopping phototypesetter at the end of the run.
−w	Wait until phototypesetter is available if busy.
−b	Report whether phototypesetter is busy or available. No text processing is done.
−a	Send a printable approximation in American Standard Code for Information Interchange (ASCII) character set of the results 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.

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 **troff** formatters, respectively).
- The table-construction preprocessor is **tbl**.
- 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**. For example, in:

  tbl file | eqn | troff −t | tc

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

3. General Information

This section describes some general principles of the **nroff** and **troff** formatters.

3.1 Form of Input

Input data consists of *text lines*, which are destined to be printed, interspersed with *control lines*, which set parameters or otherwise control subsequent processing. Control lines begin with a control character, normally 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 aesthetic reasons. Names must be followed by either a space or a newline character. Control lines with unrecognized request/macro names are ignored. There are tables in each section of this chapter that contain explanations of the request/macro names. Table 3.1.A at the end of this chapter (Page 50) is a cross reference of these tables.

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. Table 3.1.B at the end of this chapter (Page 51) itemizes escape sequences for characters, indicators, and functions.

3.2 Formatter and Device Resolution

The **troff** processor internally uses 432 units/inch, corresponding to the Wang Laboratories phototypesetter which has a horizontal resolution of 1/432 inch and a vertical resolution of 1/144 inch. It rounds horizontal/vertical numerical parameter input to the actual horizontal/vertical resolution of the typesetter.

The **nroff** 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).

3.3 Numerical Parameter Input

Both **nroff** and **troff** formatters accept numerical input with the appended scale indicators shown in the following table, 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.

TABLE 3.3. Scale Indicators in **troff** and **nroff**

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		

In **nroff** processors, 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 {8} 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.

3.4 Numerical Expressions

Wherever numerical input is expected, the following may be used:

an expression involving parentheses,
the arithmetic operators +, −, /, *, % (mod), and the
logical operators <, >, <=, >=, =, ==, & (and), : (or).

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).

3.5 Notation

Numerical parameters are indicated in this manual in two ways. A ±*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**. If no argument is specified, then the **.ps**, **.ft**, **.po**, **.vs**, **.ls**, **.ll**, **.in**, and **.lt** requests restore the previous parameter value.

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.

4. Font and Character Size Control

4.1 Fonts

Default mounted fonts 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 4 at the end of Chapter 2 in this guide. The current font, initially Times Roman, may be changed (among the mounted fonts) by use of the **.ft** request or by imbedding at any desired point either \f*x*, \f(*xx*, or \f*N* where *x* and *xx* are the name of a mounted font and *N* is a numerical font position. It is not necessary to change to the Special Font; characters on that font are automatically handled. They are accessed by their 4-character input names {4.2}. A request for a named but not mounted font is ignored.

The **troff** processor can be informed that any particular font is mounted by use of the **.fp** request. The list of known fonts is installation dependent. In the subsequent discussion of font-related requests, *F* represents either a 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**.

Font control is understood by the **nroff** formatter which normally underlines italic characters. Table 4.1 below is a summary and explanation of font control requests.

TABLE 4.1. Font Control Requests

REQUEST FORM	INITIAL VALUE	IF NO ARGUMENT	EXPLANATION
.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.
.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.
.fp *N F*	R,I,B,S	ignored	Font position. A font named *F* is mounted on position *N* (1 through 4). It is a fatal error if *F* is not known. The phototypesetter has four fonts physically mounted. Each font consists of a film strip which can be mounted on a numbered quadrant of a wheel. The default mounting sequence assumed by the **troff** formatter is R, I, B, and S on positions 1, 2, 3, and 4, respectively.
.ft *F*	Roman	previous	Change to font *F* (*F* is *x*, *xx*, 1 through 4, 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.

4.2 Character Set

The **troff** character set consists of the 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 Table 4.2.A at the end of this chapter (Page 52). The three ASCII character exceptions are mapped as follows:

TABLE 4.2.B. ASCII Character Exceptions

ASCII INPUT		PRINTED BY troff	
Character	**Name**	**Character**	**Name**
´	acute accent	'	close quote
`	grave accent	`	open quote
–	minus	-	hyphen

The characters ´, `, and – may be input by \´, \`, and \-, respectively, or by their names \(aa, \(ga, and \(mi). 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** 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.

printable characters.

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

4.3 Character Size

Character point sizes available in **troff** 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. The **.ps** request is used to change or restore the point size. Alternatively, the point size may be changed between any two characters by imbedding a \s*N* at the desired point to set the size to *N* or a \s± *N* (1 ⩽ *N* ⩽ 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. The current size is available in the .s number register. Table 4.3 below is a summary and explanation of character size requests.

Note that the **nroff** formatter ignores type size control.

TABLE 4.3. Character Size Control Requests

REQUEST FORM	INITIAL VALUE	IF NO ARGUMENT	EXPLANATION
.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	EXPLANATION
.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 are: 6, 7, 8, 9, 10, 11, 12, 14, 16, 18, 20, 22, 24, and 36. 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.
.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.

4.4 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) {8.5}. 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 Table 4.4 below. 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 phototypesetter is about 7.54 inches. The left margin begins about 1/27 inch from the edge of the 8-inch wide, continuous roll paper. Physical limitations on the **nroff** processor output are output-device dependent.

TABLE 4.4. Page Control Requests

REQUEST FORM	INITIAL VALUE*	IF NO ARGUMENT	EXPLANATION
.bp ± *N*	*N* = 1	–	Begin page. The current page is ejected and a new page is begun. If ± *N* is given, the new page number will be ± *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.
.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.
.ne *N*	–	*N* = 1*V*	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.
.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	EXPLANATION
.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.
.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 **troff** formatter the maximum (line-length) + (page-offset) is about 7.54 inches. The current page offset is available in the **.o** register.
.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|\\nRu**. Mode or relevant parameters are associated with current diversion level. The scale indicator is ignored if not specified in the request.

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

5. Text Filling, Adjusting, and Centering

5.1 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 an *unpaddable space* by using a 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.

To obtain a specific break in a line when filling is in effect, a **\p** sequence may be imbedded in or attached to a word to cause a break at the end of that word and have the resulting output of the line containing that word spread out to fill the current line length.

A text input line that happens to begin with a control character (such as a period) can be made to be interpreted as the actual character itself 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.

Table 5.1 below is a summary and explanation of filling and adjusting requests.

TABLE 5.1. Text Filling, Adjusting, And Centering Requests

REQUEST FORM	INITIAL VALUE	IF NO ARGUMENT	EXPLANATION
.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.
.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.
.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.
.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.
.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	EXPLANATION
.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.

5.1.1 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 or partial word will be forced out.

5.2 Centering

A line or lines of text can be centered horizontally with the **.ce** request. The basis for centering is the line length minus the indent {7}.

Table 5.1 above contains an explanation of the centering request.

6. Vertical Spacing

A summary and explanation of vertical spacing requests can be found in Table 6.3.

6.1 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 **troff** formatter and to the output device resolution in the **nroff** formatter. 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** (line spacing) request.

6.2 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 imbedded 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.

6.3 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.

Table 6.3 below is a summary and explanation of vertical spacing requests.

TABLE 6.3. Vertical Spacing Requests

REQUEST FORM	INITIAL VALUE*	IF NO ARGUMENT	EXPLANATION
.ls N	$N = 1$	previous	Line spacing set to $\pm N$. Output $N-1$ blank lines (Vs) 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	EXPLANATION
.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.
.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.
.rs	–	–	Restore spacing. The no-space mode (**.ns**) is turned off. Mode or relevant parameters are associated with current diversion level.
.sp *N*	–	*N* = 1*V*	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.
.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.
.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.
Blank line	–	–	This condition causes a break and output of a blank line (just as does **.sp 1**).

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

7. 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** (temporary indent) 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** {15}. Table 7 is a summary and explanation of line length and indenting requests.

TABLE 7. Line Length and Indenting Requests

REQUEST FORM	INITIAL VALUE	IF NO ARGUMENT	EXPLANATION
.in ± *N*	*N* = 0	previous	Indent. The indent is set to ± *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.
.ll ± *N*	6.5 in	previous	Line length. The line length is set to ± *N*. In the **troff** 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.
.ti ± *N*	–	ignored	Temporary indent. The next output text line will be indented a distance ± *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.

8. Macros, Strings, Diversions, and Position Traps

A summary and explanation of macro, string, diversion, and position trap requests can be found in Table 8.5.

8.1 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 within text.

8.2 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 {8.1}.
- Arguments indicated by **\$** are interpolated.
- Concealed newline characters indicated by \<**newline**> are eliminated.
- Comments indicated by \" are eliminated {11.7}.
- **\t** and **\a** are interpreted as ASCII horizontal tab and start of heading (SOH), respectively {10.1}.

- \\ 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** which will be interpreted as a number register indicator when the macro or string is reread.

8.3 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 imbedded space characters. Pairs of double-quotes may be imbedded 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 ($1 \leqslant N \leqslant 9$). 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,
- from within a string because string referencing is implemented as an input-level pushdown, or 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.

8.4 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 imbed in the diversion the appropriate **.cs** or **.bd** request with the transparent mechanism described in section 11.6.

Diversions may be nested and certain parameters and registers are associated with the current diversion level (the top non-diversion level may be thought of as 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).

8.5 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.

Table 8.5 below is a summary and explanation of macro, string, diversion, and position trap requests.

TABLE 8.5. Macros, Strings, Diversions, and Position Traps Requests

REQUEST FORM	INITIAL VALUE	IF NO ARGUMENT	EXPLANATION
.am *xx yy*	–	*yy* = ..	Append to macro *xx* (append version of **.de**).
.as *xx string*	–	ignored	Append *string* to string *xx* (append version of **.ds**).

REQUEST FORM	INITIAL VALUE	IF NO ARGUMENT	EXPLANATION
.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.
.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.
.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 "..".
.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.
.ds *xx string*	–	ignored	Define a string *xx* containing *string*. Any initial double-quote in *string* is stripped to permit initial blanks.
.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	EXPLANATION
.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.
.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.
.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.
.rn *xx yy*	–	ignored	Rename. Rename request, macro, or string from *xx* to *yy*. If *yy* exists, it is first removed.
.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.

9. Number Registers

A variety of predefined number registers are available to the user and are listed at the end of this chapter in Table 9.A (Page 55). 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, a number register can be read, written, automatically incremented or decremented, and interpolated into the input in a variety of formats. These read-only number registers are listed at the end of this chapter in Table 9.B (Page 56). 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*, the following access sequences have the effect shown:

TABLE 9.C. 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*

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.

Table 9.D below is a summary and explanation of number registers requests.

TABLE 9.D. Number Registers Requests

REQUEST FORM	INITIAL VALUE	IF NO ARGUMENT	EXPLANATION
.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.
.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.
.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.

10. Tabs, Leaders, and Fields

A summary and explanation of tab, leader, and field requests can be found in Table 10.2.

10.1 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 the following table:

- *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*.

TABLE 10.1. Internal Tab Stops

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*

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.

10.2 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.

Table 10.2 below is a summary and explanation of tab, leader, and field requests.

TABLE 10.2. Tab, Leader, and Field Requests

REQUEST FORM	INITIAL VALUE	IF NO ARGUMENT	EXPLANATION
.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.
.lc *c*	.	none	Leader repetition character becomes *c* or is removed specifying motion. Relevant parameters are a part of the current environment.
.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.
.tc *c*	none	none	Tab repetition character becomes *c* or is removed specifying motion. Relevant parameters are a part of the current environment.

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

11. Input/Output Conventions and Character Translations

11.1 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 Table 3.1.B. 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 Table 11.5.

11.2 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 Table 11.5.

11.3 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 as a form of line drawing and, as a generalized overstriking function, is described in section 13.

The **nroff** processor underlines characters automatically in the underline font, specifiable with the **.uf** request. The underline font is normally on font position 2 (Times Italic). 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 Table 11.5.

11.4 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 Table 11.5.

11.5 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 Table 11.5 below is a summary and explanation of the output translation request.

TABLE 11.5. Input and Output Conventions and Character Translation Requests

REQUEST FORM	INITIAL VALUE	IF NO ARGUMENT	EXPLANATION
.cc *c*	.	.	Set control character to **c** or reset to ".". Relevant parameters are a part of the current environment.
.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.
.c2 *c*	'	'	Set no-break control character to **c** or reset to "'". Relevant parameters are a part of the current environment.
.ec *c*	\	\	Set escape character to \ or to *c* if given.
.eo	on	—	Turn escape character mechanism off.
.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.
.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.
.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	EXPLANATION
.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.

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

11.6 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 imbed control lines in a macro created by a diversion.

11.7 Comments and Concealed Newline Characters

An uncomfortably long input line that must stay 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 imbedded 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 .\".

12. Local Horizontal/Vertical Motion and Width Function

12.1 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 Table 12.1 at the end of this chapter (Page 57).

As an example, E^2 is generated by the sequence

```
E\v'-.5'\s-4\&2\s0\v'.5'
```

12.2 Width Function

The width function **w**'*string*' generates the numerical width of *string* (in basic units). Size and font changes may be imbedded 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 respectively to the highest and lowest extent of *string* relative to the baseline; then, for example, the total height of the string is **\n(stu-\n(sbu**. In 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 the character **e**)
- **1** means that at least one character has a descender (like the character **y**)
- **2** means that at least one character is tall (like the character **H**)
- **3** means that both tall characters and characters with descenders are present.

12.3 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\fIword\fR\h'|\nxu+2u'\fIword\fR
```

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

word

13. Overstrike, Zero-Width, Bracket, and Line Drawing Functions

13.1 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. For example:

\o'e\'' produces é
\o'>/' produces ⊁

13.2 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 (□).

13.3 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⎤
⎣ ⎦

13.4 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

words in a box

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'       \"draw box
.fi
..
```

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

14. 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 imbedded 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. Table 14 below is a summary and explanation of hyphenation requests.

TABLE 14. Hyphenation Requests

REQUEST FORM	INITIAL VALUE	IF NO ARGUMENT	EXPLANATION
.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.
.hw *word1...*	–	ignored	Exception words. Hyphenation points in words are specified with imbedded 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.
.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.
.nh	no hyphen	–	No hyphenation. Automatic hyphenation is turned off. Relevant parameters are a part of the current environment.

15. 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. Table 15 below is a summary and explanation of 3-part title requests.

TABLE 15. Three-Part Titles Requests

REQUEST FORM	INITIAL VALUE	IF NO ARGUMENT	EXPLANATION
.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.
.pc *c*	%	off	Page number character set to *c* or removed. The page number register remains %.

REQUEST FORM	INITIAL VALUE	IF NO ARGUMENT	EXPLANATION
.tl'*left*'*center*'*right*'	–		Three-part title. The strings *left*, *center*, and *right* are respectively left-adjusted, centered, and right-adjusted in the current title length. Any of the strings may be empty, and overlapping is permitted. If the page number character (initially %) is found within any of the fields, it is replaced by the current page number having the format assigned to register %. Any character may be used as the string delimiter.

16. 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). Table 16 is a summary and explanation of output line numbering requests.

TABLE 16. Output Line Numbering Requests

REQUEST FORM	INITIAL VALUE	IF NO ARGUMENT	EXPLANATION
.nm ±*N M S I*	–	off	Line number mode. If ±*N* is given, line numbering is turned on, and the next output line is numbered ±*N*. Default values are *M*=1, *S*=1, and *I*=0. Parameters corresponding to missing arguments are unaffected; a non-numeric argument is considered missing. In the absence of all arguments, numbering is turned off, and the next line number is preserved for possible further use in number register **ln**. Relevant parameters are a part of the current environment.
.nn *N*	–	*N* = 1	Next *N* lines are not numbered. Relevant parameters are a part of the current environment.

Figure 16 is an example of output line numbering. Paragraph portions are numbered with *M*= 2.

```
    Automatic sequence numbering of output lines may be requested
 2  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
 4  four digit-spaces and otherwise retain their line length. A reduc-
    tion in line length (such as .ll -\w'0000'u in this example) may
 6  be desired to keep the right margin aligned with an earlier margin.

    Blank lines, other vertical spaces, and lines generated by .tl are not
 8  numbered. Numbering can be temporarily suspended with .nn or
    with a .nm followed by a later .nm +0.

10  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
12  only those line numbers that are multiples of some number M are
    to be printed (the others will appear as blank number fields). This
14  example uses the multiple of 2.
```

Figure 16. Output Line Numbering

- **.ll −\w'0000'u** was placed at the beginning to keep the right

margin aligned
- **.nm 1 2** was placed at the beginning
- **.nm +0** was placed in front of the second and third paragraphs
- **.nm** was placed at the end
- **.ll +\w'0000'u** was placed at the end to return to the original line length

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.

17. Conditional Acceptance of Input

In Table 17 below, which is a summary and explanation of conditional acceptance requests:

- *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.

TABLE 17.A. Conditional Acceptance of Input Requests

REQUEST FORM	EXPLANATION
.el *anything*	The "else" portion of "if-else".
.ie *c anything*	The "if" portion of "if-else". The *c* can be any of the forms acceptable with the .if request.
.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.
.if *!c anything*	If condition *c* false, accept *anything*.
.if *N anything*	If expression $N > 0$, accept *anything*. The scale indicator is ignored if not specified in the request.

REQUEST FORM	EXPLANATION
.if *!N anything*	If expression $N \leq 0$ accept *anything*. The scale indicator is ignored if not specified in the request.
.if *'string1' string2' anything*	If *string1* is identical to *string2*, accept *anything*.
.if *!'string1' string2' anything*	If *string1* is not identical to *string2*, accept *anything*.

TABLE 17.B. Built-in Condition Names

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

If 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 \}.

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.

18. Environment Switching

A number of parameters that control text processing are gathered together into an environment, which can be switched by the user. Environment parameters are those associated with some requests. The request tables in this chapter 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. Table 18 is a summary and explanation of the environment switching request.

TABLE 18. Environment Switching Request

REQUEST FORM	INITIAL VALUE	IF NO ARGUMENT	EXPLANATION
.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.

19. 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 operating 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. Table 19 below is a summary and explanation of insertions from the standard input requests.

TABLE 19. Insertions from Standard Input Requests

REQUEST FORM	INITIAL VALUE	IF NO ARGUMENT	EXPLANATION
.ex	–	–	Exit from the **nroff/troff** formatter. Text processing is terminated exactly as if all input had ended.
.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*.

20. Input/Output File Switching

Table 20 below is a summary and explanation of input/output file switching requests.

TABLE 20. Input/Output File Switching Requests

REQUEST FORM	INITIAL VALUE	IF NO ARGUMENT	EXPLANATION
.nx *filename*	–	end-of-file	Next file is *filename*. The current file is considered ended, and the input is immediately switched to *filename*.
.pi *program*	–	–	Pipe output to *program* (**nroff** formatter only). This request must occur before any printing occurs. No arguments are transmitted to *program*.
.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.

21. Miscellaneous

Table 21 below is a summary and explanation of miscellaneous requests.

TABLE 21. Miscellaneous Requests

REQUEST FORM	INITIAL VALUE	IF NO ARGUMENT	EXPLANATION
.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.
.fl	–	–	Flush output buffer. Used in interactive debugging to force output. The request causes a break.
.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.
.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.
.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.
.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.

22. Output and Error Messages

Output from **.tm**, **.pm**, and prompt from **.rd**, as well as various error messages are written onto the UNIX operating system standard message output. The latter is different from the standard output, when compared to the **nroff** formatted 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 a * (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. Table 22 is a summary and explanation of output and error messages requests.

TABLE 22. Output and Error Messages Request

REQUEST FORM	INITIAL VALUE	IF NO ARGUMENT	EXPLANATION
.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.

23. Compacted Macros

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**.

23.1 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
        .
        .
        .
```

23.2 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 −kmac mac
```

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

23.3 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
```

or

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

23.4 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
```

24. Reference Tables

TABLE 3.1.A. Cross Reference Request to Table & Page Number

REQUEST NAME	TABLE NUMBER	PAGE NUMBER	REQUEST NAME	TABLE NUMBER	PAGE NUMBER
ab	22	47	**ls**	6.3	19
ad	5.1	17	**lt**	15	39
af	9.D	29	**mc**	21	46
am	8.5	25	**mk**	4.4	14
as	8.5	25	**na**	5.1	17
bd	4.1	10	**ne**	4.4	14
bp	4.4	14	**nf**	5.1	18
br	5.1	17	**nh**	14	39
c2	11.5	33	**nm**	16	41
cc	11.5	33	**nn**	16	41
ce	5.1	17	**nr**	9.D	29
ch	8.5	26	**ns**	6.3	20
co	21	46	**nx**	20	45
cs	4.3	12	**os**	6.3	20
cu	11.5	33	**pc**	15	39
da	8.5	26	**pi**	20	45
de	8.5	26	**pl**	4.4	14
di	8.5	26	**pm**	21	46
ds	8.5	26	**pn**	4.4	15
dt	8.5	26	**po**	4.4	15
ec	11.5	33	**ps**	4.3	13
el	17.A	42	**rd**	19	45
em	8.5	27	**rm**	8.5	27
eo	11.5	33	**rn**	8.5	27
ev	18	44	**rr**	9.D	29
ex	19	45	**rs**	6.3	20
fc	10.2	31	**rt**	4.4	15
fi	5.1	17	**so**	20	45
fl	21	46	**sp**	6.3	20
fp	4.1	10	**ss**	4.3	13
ft	4.1	10	**sv**	6.3	20
hc	14	38	**ta**	10.2	31
hw	14	39	**tc**	10.2	31
hy	14	39	**ti**	7	21
ie	17.A	42	**tl**	15	40
if	17.A	42	**tm**	21	46
ig	21	46	**tr**	11.5	33
in	7	21	**uf**	11.5	33
it	8.5	27	**ul**	11.5	34
lc	10.2	31	**vs**	6.3	20
lg	11.5	33	**wh**	8.5	27
ll	7	21			

TABLE 3.1.B. Escape Sequences for Characters, Indicators, and Functions

ESCAPE SEQUENCE	MEANING
\\	\ (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 **nroff**)
\^	1/12 em half-narrow space character (zero width in **nroff**)
\&	Nonprinting zero width character
\!	Transparent line indicator
\"	Beginning of comment
\$*N*	Interpolate argument ($1 \leqslant N \leqslant 9$)
\%	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 **nroff**)
\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
\j*x*, \j(*xx*	Mark current horizontal output position in register *x* or *xx*
\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 **nroff**)
\s*N*, \s±*N*	Point-size change function
\t	Noninterpreted horizontal tab
\u	Reverse (up) ½ em vertical motion (½ line in **nroff**)
\v'*N*'	Local vertical motion
\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

Note: Escape sequences \\, \., \", \$, *, \a, \n, \t, \<newline> are interpreted in copy mode.

TABLE 4.2.A. Naming Conventions for Non-ASCII Characters

Non-ASCII characters and *minus* on the standard fonts.					
CHAR	INPUT NAME	CHARACTER NAME	CHAR	INPUT NAME	CHARACTER NAME
'	'	close quote	fi	\(fi	fi
`	`	open quote	fl	\(fl	fl
—	\(em	¾ Em dash	ff	\(ff	ff
-	-	hyphen or	ffi	\(Fi	ffi
-	\(hy	hyphen	ffl	\(Fl	ffl
—	\-	current font minus	°	\(de	degree
•	\(bu	bullet	†	\(dg	dagger
□	\(sq	square	′	\(fm	foot mark
_	\(ru	rule	¢	\(ct	cent sign
¼	\(14	one-fourth	®	\(rg	registered
½	\(12	one-half	©	\(co	copyright
¾	\(34	three-fourths			

TABLE 4.2.A. Naming Conventions for Non-ASCII Characters (continued)

Non-ASCII Greek characters on the special font.					
CHAR	INPUT NAME	CHARACTER NAME	CHAR	INPUT NAME	CHARACTER NAME
A	\(*A	Alpha†	α	\(*a	alpha
B	\(*B	Beta†	β	\(*b	beta
Γ	\(*G	Gamma	γ	\(*g	gamma
Δ	\(*D	Delta	δ	\(*d	delta
E	\(*E	Epsilon†	ϵ	\(*e	epsilon
Z	\(*Z	Zeta†	ζ	\(*z	zeta
H	\(*Y	Eta†	η	\(*y	eta
Θ	\(*H	Theta	θ	\(*h	theta
I	\(*I	Iota†	ι	\(*i	iota
K	\(*K	Kappa†	κ	\(*k	kappa
Λ	\(*L	Lambda	λ	\(*l	lambda
M	\(*M	Mu†	μ	\(*m	mu
N	\(*N	Nu†	ν	\(*n	nu
Ξ	\(*C	Xi	ξ	\(*c	xi
O	\(*O	Omicron†	o	\(*o	omicron
Π	\(*P	Pi	π	\(*p	pi
P	\(*R	Rho†	ρ	\(*r	rho
Σ	\(*S	Sigma	σ	\(*s	sigma
			ς	\(ts	terminal sigma
T	\(*T	Tau†	τ	\(*t	tau
Υ	\(*U	Upsilon	υ	\(*u	upsilon
Φ	\(*F	Phi	ϕ	\(*f	phi
X	\(*X	Chi†	χ	\(*x	chi
Ψ	\(*Q	Psi	ψ	\(*q	psi
Ω	\(*W	Omega	ω	\(*w	omega

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

TABLE 4.2.A. Naming Conventions for Non-ASCII Characters (continued)

Non-ASCII characters and ´, `, _, +, −, =, * on special font.					
CHAR	INPUT NAME	CHARACTER NAME	CHAR	INPUT NAME	CHARACTER NAME
+	\(pl	math plus	*	\(**	math star
−	\(mi	math minus	|	\(or	or
±	\(+−	plus-minus	/	\(sl	slash
×	\(mu	multiply	§	\(sc	section
÷	\(di	divide	´	\(aa	acute accent
=	\(eq	math equals	`	\(ga	grave accent
≥	\(>=	greater than or equal	_	\(ul	underrule
≤	\(<=	less than or equal	→	\(−>	right arrow
≡	\(==	identically equal	←	\(<−	left arrow
≅	\(=	approximately equal	↑	\(ua	up arrow
~	\(ap	approximates	↓	\(da	down arrow
≠	\(!=	not equal	‡	\(dd	double dagger
√	\(sr	square root	Ⓐ	\(bs	Bell System logo
	\(rn	root en extender	☚	\(lh	left hand
∪	\(cu	cup (union)	☛	\(rh	right hand
∩	\(ca	cap (intersection)	|	\(br	box vertical rule
⊂	\(sb	subset of	○	\(ci	circle
⊃	\(sp	superset of	|	\(bv	bold vertical
⊆	\(ib	improper subset	⌈	\(lc	left ceiling (bracket)
⊇	\(ip	improper superset	⌉	\(rc	right ceiling
∈	\(mo	member of	⌊	\(lf	left floor
∅	\(es	empty set	⌋	\(rf	right floor
∞	\(if	infinity	{	\(lt	left top (brace)
∂	\(pd	partial derivative	}	\(rt	right top
∇	\(gr	gradient	⎩	\(lb	left bottom
∫	\(is	integral sign	⎭	\(rb	right bottom
∝	\(pt	proportional to	⎨	\(lk	left center
¬	\(no	not	⎬	\(rk	right center

TABLE 9.A. Predefined General Number Registers

REGISTER NAME	DESCRIPTION
%	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.
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.
.R	Number of number registers that remain available for use.
.b	Emboldening factor of the current font.

TABLE 9.B. Predefined Read-Only Number Registers

REGISTER NAME	DESCRIPTION
.$	Number of arguments available at the current macro level.
.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*'.
.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.
.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.

TABLE 12.1. Local Motions

Vertical Local Motion

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

Horizontal Local Motion

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

Chapter 4: TABLE FORMATTING PROGRAM

CONTENTS

LIST OF FIGURES

Chapter 4

TABLE FORMATTING PROGRAM

1. Introduction

The **tbl** program is a document formatting preprocessor for the **troff** and **nroff** formatters which makes fairly complex tables easy to specify and enter. Tables consist of columns which may be independently centered, right-adjusted, left-adjusted, or aligned by decimal points. Headings may be placed over single columns or groups of columns. A table entry may contain equations or consist of several rows of text. Horizontal or vertical lines may be drawn as desired in the table, and any table or element may be enclosed in a box.

A description of a table is put by the **tbl** program into an **nroff/troff** formatter list of requests that prints the table. The **tbl** program isolates and processes the portion of a job that contains **tbl** commands and leaves the remainder for other programs. Thus, **tbl** may be used with the equation formatting program (**eqn**) and/or various formatter layout macro packages without function duplication.

2. General Usage

On the UNIX operating system, the **tbl** program can be run on a simple table with the command

tbl *filename* | **nroff**

or

tbl *filename* | **troff**

Any of the usual options may be used to the **nroff** and **troff** formatters. If a file name is " − ", the standard input is read at that point.

When there are several input files containing tables, equations, and **mm** macro requests, the correct sequence would be

tbl file1 file2 file3 | eqn | troff −mm

Usage of the **nroff** formatter is similar to that of **troff**, but only TELETYPE® Model 37 and Diablo-mechanism (DASI or GSI) terminals can print vertical lines or boxed tables. For the convenience of users

employing line printers without adequate driving tables or post-filters, there is a special **−TX** command-line option to **tbl** which produces output that does not have fractional line motions. The only other command-line options recognized by **tbl** are **−ms** and **−mm**. They are turned into commands to fetch the corresponding macro files. It is usually more convenient to place these arguments on the **nroff/troff** formatter part of the command line, but they are accepted by **tbl** as well.

The **tbl** program accepts up to 35 columns; the actual number that can be processed may be smaller depending on availability of **nroff/troff** formatter number registers. Number register names used by **tbl** must be avoided within tables. These include 2-digit numbers from 31 to 99 and strings of the form *4x*, *5x*, *#x*, *x+*, *x|*, *^x*, and *x−*, where *x* is any lowercase letter. The names *##*, *#−*, and *#^* are also used in certain circumstances. To conserve register names, the **n** and **a** key letters (key letters are introduced in section 5.2) share a register. Hence, the restriction that they may not be used in the same column.

3. Multipage Tables

As an aid in writing layout macros, **tbl** defines a number register TW which is the table width. The TW number register is defined by the time that the **.TE** macro is invoked and may be used in the expansion of that macro. More importantly, to assist in laying out multipage boxed tables, the macro **T#** is defined to produce the bottom lines and side lines of a boxed table and then be invoked at its end. By use of this macro in the page footer, a multipage table can be boxed.

In particular, the **ms** and **mm** macros can be used to print a multipage boxed table with a repeated heading by giving the argument **H** to the **.TS** macro. If the table start macro is written

```
.TS H
```

then a line of the form

```
.TH
```

must be given in the table after any table heading (or at the start if none). Material up to the **.TH** is placed at the top of each page of the table. The remaining lines in the table are placed on several pages as required. This is not a feature of **tbl** but of the **ms** and **mm** macros, and therefore, must be used in conjunction with either of these macro packages.

4. Usage with EQN

When both **tbl** and **eqn** programs operate on the same file, **tbl** should be called first. It is usually faster to execute **tbl** first since **eqn** normally produces a larger expansion of the input. However, if there are equations within tables, **tbl** must be executed first or the output will be scrambled. If there are no equations within tables, either sequence works.

Use of equations in **n**-style (numerical) columns should be avoided since **tbl** attempts to split numerical format items into two parts. The **delim** (*xy*) global option in the table prevents splitting numerical columns within delimiters. For example, if the **eqn** delimiters are

```
1245 $\(+- 16$
```

to be divided after 1245, not after 16.

5. Input Requests

Input to **tbl** is text for a table preceded by a **.TS** (table start) request and followed by a **.TE** (table end) request. The **tbl** program processes the tables, generates formatting requests, and leaves the text unchanged. The **.TS** and **.TE** lines are copied so that **troff** formatter layout macros (such as memorandum macros) can use these lines as delimiters. Arguments on the .TS or **.TE** lines are copied, but otherwise ignored, and may be used by document layout macro requests.

The general format of the input is

preceding text
.TS
table
.TE
more text
.TS
table
.TE
more text

The format of each table is

.TS
options;
format.
data
.TE

Each table is independent and contains:

- Global options.
- A format line describing individual columns and rows of the table.
- Data to be printed.

The format section and data are always required but global options are not.

5.1 Global Options

There may be a single line of options affecting the whole table. If present, this line must immediately follow the **.TS** line and must contain a list of option names separated by spaces, tabs, or commas and must be terminated by a semicolon. Allowable options are:

- **center** — center entire table (default is left-adjust)
- **expand** — make table as wide as current line length
- **box** — enclose table in a box
- **allbox** — enclose each item of table in a box
- **doublebox** — enclose table in two boxes
- **tab** (*x*) — separate data items by using *x* instead of tab
- **linesize** (*n*) — set lines or rules (e.g., from **box**) in *n*-point type
- **delim** (*xy*) — recognize *x* and *y* as **eqn** delimiters.

The **tbl** program tries to keep boxed tables on one page by issuing appropriate **.ne** (need) requests. These requests are calculated from the number of lines in the tables. If there are spacing requests embedded in the input, the **.he** requests may be inaccurate. Normal **troff** formatter procedures, such as keep-release macros or specifying the **.ne** *N* request, are used in that case. If a multipage boxed table is required, macros designed for this purpose (**.TS H** and **.TH**) should be used.

5.2 Format Key Letters

The format section of the table specifies the layout of the columns. Each line in the format section corresponds to one line of table data, except for the last format line which corresponds to all following data lines up to the table end (**.TE**) request or a **.T&** request line {6}. Each format line contains a *key letter* for each column of the table. Key letters may be separated by spaces or tabs for readability purposes. Key letters are:

L or **l**	Indicates a left-adjusted column entry.
R or **r**	Indicates a right-adjusted column entry.
C or **c**	Indicates a centered column entry.
N or **n**	Indicates a numerical column entry. Numerical entries are aligned so that the digits of numbers line up (it recognizes decimal points).
A or **a**	Indicates an alphabetic subcolumn. All corresponding entries are aligned on the left and positioned so that the widest entry is centered within the column.
S or **s**	Indicates a spanned heading. The entry from the previous column continues across this column (not allowed for the first column of the table).
^	Indicates a vertically spanned heading. The entry from the previous row continues down through this row (not allowed for the first row of the table).

5.2.1 Format Layout

The end of the format section is indicated by a period. The layout of key letters in the format section resembles the layout of the actual data in the table. Thus, a simple 3-column format might appear as

```
c s s
l n n.
```

The first line of the table contains a centered heading spanned across all three columns. Each remaining line contains a left-adjusted item in the first column followed by two columns of numerical data. A sample table in this format is:

Overall Title

Item-a	34.22	9.1
Item-b	12.65	.02
Item-c	23	5.8
Total	69.87	14.92

Instead of listing the format of successive lines of a table on consecutive lines of the format section, successive line formats may be given on the same line, separated by commas. The format for the above example could be written:

```
c s s, l n n.
```

Spaces between the key letters are not necessary but are visually helpful in setting up or changing a table format.

5.2.2 Numerical Column

When numerical column alignment (**n**) is specified, a location for the decimal point is sought. The rightmost dot (.) adjacent to a digit is used as a decimal point. If there is no dot adjoining a digit, the rightmost digit is used as a units digit. If no alignment is indicated, the item is centered in the column. However, the special nonprinting character string \& may be used to override dots and digits or to align alphabetic data. This string lines up where a dot normally would (the \& disappears from the final output). In the following example, items shown in the INPUT column will be aligned (in a numerical column) as shown in the OUTPUT column.

INPUT:	OUTPUT:
13	13
4.2	4.2
26.4.12	26.4.12
abcdefg	abcdefg
abcdefg\&	abcdefg
43\&3.22	433.22
749.12	749.12

If numerical data is used in the same column with wider l- or r-type table entries, the widest number is centered relative to the wider l or r items. Alignment within the numerical items is preserved. For example, the input:


```
.TS
l l
n n.
short⊕ longest entry
13⊕ 13
42,347.99⊕ 42,347.99
0.5⊕ 0.5
.TE
```

will output:

short	longest entry
13	13
42,347.99	42,347.99
0.5	0.5

This is similar to the behavior of **a** type data. Alphabetic subcolumns (requested by the **a** key letter) are always slightly indented relative to **l** items. If necessary, the column width is increased to force this. This is not true for **n** type entries.

Note: The **n** and **a** items should not be used in the same column.

5.2.3 Key Letter Features

Additional features of the key letter system are:

- *Horizontal lines* — A key letter may be replaced by underscore (_) to indicate a horizontal line in place of the column entry or equal sign (=) to indicate a double horizontal line. If an adjacent column contains a horizontal line or if there are vertical lines adjoining this column, the horizontal line is extended to meet nearby lines. If any data entry is provided for this column, it is ignored and a warning message is printed.
- *Vertical lines* — A vertical bar (|) placed between column key letters will cause a vertical line between the corresponding columns of the table. A vertical bar to the left of the first key letter or to the right of the last one produces a line at the edge of the table. If two vertical bars appear between key letters, a double vertical line is drawn.
- *Space between columns* — A number may follow the key letter indicating the amount of separation between this column and the next column. The number specifies the separation in *ens*. One *en*

is about the width of the letter "n". More precisely, an *en* is the number of points (1 point = 1/72 inch) equal to half the current type size. If the *expand* option is used, these numbers are multiplied by a constant such that the table is as wide as the current line length. The default column separation number is 3. If the separation is changed, the worst case (largest space requested) governs.

- *Vertical spanning* — Vertically spanned items extending over several rows of the table are centered in their vertical range. This is done by using the ^ symbol as a key letter in the columns the item is to be spanned. If a key letter is followed by **t** or **T**, any corresponding vertically spanned item will begin at the top line of its range.
- *Font changes* — A key letter followed by a string containing a font name (such as **R**, **I**, or **B**) or number (such as **1**, **2**, or **3**) preceded by the letter **f** or **F** indicates that the corresponding column should be in a different font from the default font (usually Roman). All font names are one or two letters. A 1-letter font name should be separated from whatever follows by a space or tab. The single letters **B**, **b**, **I**, and **i** are shorter synonyms for **fB** and **fI**. Font-change requests given within the table data override these specifications.
- *Point size changes* — A key letter followed by **p** or **P** and a number indicates the point size of the corresponding table entries. If the number is a signed (+ or −) digit, it is taken as an increment or decrement from the current point size. If both a point size and a column separation value are given, one or more blanks must separate them.
- *Vertical spacing changes* — A key letter followed by **v** or **V** and a number indicates the vertical line spacing used within a multiline table entry. The number may be a signed (+ or −) digit, in which case it is taken as an increment or decrement from the current vertical spacing. A column separation space value must be separated by blanks or some other specification from a vertical spacing request. This request has no effect unless the corresponding table entry is a block of text.
- *Column width indication* — A key letter followed by **w** or **W** and a width value in parentheses indicates minimum column width. If the largest element in the column is not as wide as the width value given after the **w**, the largest element is assumed to be that wide. If the largest element in the column is wider than the specified value, its width is used. The width is also used as a default line length for text blocks. Normal **troff** formatter units can be used to scale the width value. The default value is *ens* if none are used. If the width specification is a unitless integer, the parentheses may be omitted. If another width value is given in a column, the last one controls the width.
- *Equal-width columns* — A key letter followed by **e** or **E** indicates equal-width columns. All columns whose key letters are followed by **e** or **E** are made the same width. This permits a group of regularly spaced columns.
- *Staggered columns* - A key letter followed by **u** or **U** indicates that the corresponding entry is to be moved up one-half line. This makes it easy to have a column of differences between numbers in an adjoining column. **Note:** The *allbox* option does not work with staggered columns.
- *Zero-width item* — A key letter followed by **z** or **Z** indicates that the corresponding data item is to be ignored in calculating column widths. This may be useful in allowing a long heading to run across adjacent columns where a spanned heading would be inappropriate.
- *Default* — Column descriptors missing from the end of a format line are assumed to be **l**. The longest line in the format section, however, defines the number of columns in the table. Extra columns in the data are ignored.

The order of the features is immaterial. They need not be separated by spaces except as indicated to avoid ambiguities involving point size and font changes. Thus, a numerical column entry in italic font and 12-point type with a minimum width of 2.5 inches and separated by 6 ens from the next column could be specified as

```
np12w(2.5i)fI 6
```

5.3 Table Data

Data for the table is input after the format section. Each table line is typed as one line of data. Very long input lines can be broken. Any line whose last character is a backslash (\) is combined with the following line; i.e., the backslash vanishes. Data for each column is separated by a tab or by whatever character has been specified in the **tab** *x* global

option {4.1}. There are a few special cases of data entries:

- **troff** *requests within tables* — An input line beginning with a dot and followed by anything but a number (*.xx*) is assumed to be a request to the formatter and is passed through unchanged retaining its position in the table. For example, a space within a table may be produced with the **.sp** request in the data.
- *Full-width horizontal lines* — An input line containing only the _ (underscore) character or = (equal sign) is taken to be a single or double line, respectively, extending the full width of the table.
- *Single-column horizontal lines* — An input table entry containing only the _ character or the = is taken to be a single or double line, respectively, extending the full width of the column. Such lines are extended to meet horizontal or vertical lines adjoining this column. To actually obtain these characters explicitly in a column, they should be preceded by a **\&** or followed by a space before the usual tab or newline character.
- *Short horizontal lines* — An input table entry containing only the string _ is assumed to be a single line as wide as the contents of the column. It is not extended to meet adjoining lines.
- *Repeated characters* — An input table entry containing only a string of the form **\R***x*, where *x* is any character, is replaced by repetitions of the character *x* as wide as data in the column. The sequence is not extended to meet adjoining columns.
- *Vertically spanned items* — An input table entry containing only the \^ character string indicates that the table entry immediately above spans downward over this row. It is equivalent to a table format key letter of ^.
- *Text blocks* — In order to include a block of text as a table entry, precede it by **T{** and input text on a new line, then follow the text by a new line and **T}**. Thus, the sequence

```
... T{
block of
text
T} ...
```

  is the way to enter something as a single entry that cannot conveniently be typed as a simple string between tabs. The **}T** (begin delimeter) must be followed by a new line. The **T}** (end delimeter) must begin a new line; however, additional columns of data may follow after a tab on the same line. Text blocks are pulled out from the table, processed separately by the formatter, and replaced in the table as a solid block.

Various limits in the **troff** program are likely to be exceeded if 30 or more text blocks are used in a table. This produces diagnostic messages such as "too many string/macro names" or "too many number registers".

If no line length is specified in the block of text or in the table format, the default used is

$$L \times C / (N + 1)$$

where L is the current line length, C is the number of table columns spanned by the text, and N is the total number of columns in the table.

Other parameters (point size, font, etc.) used in typesetting the text block are:

(a) those in effect at the beginning of the table (including the effect of the **.TS** macro)

(b) any table format specifications of size, spacing, and font using the **p**, **v**, and **f** modifiers to the column key letters

(c) **troff** requests within the text block itself (requests within the table data but not within the text block do not affect that block).

Although any number of lines may be present in a table, only the first 200 lines are used in setting up the table. A multipage table may be arranged as several single-page tables if this proves to be a problem.

When calculating column widths, all table entries are assumed to be in the font and size being used when the **.TS** request was encountered. This is true except for font and size changes indicated in the table format section or within the table data (as in the entry **\s+3Data\s0**). Because arbitrary **troff** requests may be sprinkled in a table, care must be taken to avoid confusing width calculations.

6. Additional Command Lines

To change the format of a table after many similar lines, as with subheadings or summarizations, the **.T&** (table continue) request is used to change column parameters. It is not recognized after the first 200 lines of a table. It is not possible to format changes to the number of columns, the space between columns, the global options such as *box*, or the selection of columns to be made equal in width. An example of such a table input is:

```
.TS
box expand;
c s s
l l l.
data
.T&
l s s
c c c.
data
.T&
l l l.
data
.TE
```

Using this procedure, each data line can be close to its corresponding format line.

7. Examples

Figures 7.A through 7.F are included to show input and output information that illustrate the basic concepts of the **tbl** program. The ① symbol in the input data represents a tab character. Although each figure has a title that indicates an option or feature, other examples of use may be gleaned from them. For instance, Fig. 7.E shows the use of additional request lines and also specifies bold type print in the format area.

INPUT:

```
.TS
box expand;
c c c
l l n.
Name① Department① Extension
.sp
Susan Thompson① Marketing① 224
Diane Spencer① Technical Support① 360
Ramona Goodman① Documentation① 114
Terry O'Malley① Programming① 101
Willette Taylor① Accounting① 600
.TE
```

OUTPUT:

Name	Department	Extension
Susan Thompson	Marketing	224
Diane Spencer	Technical Support	360
Ramona Goodman	Documentation	114
Terry O'Malley	Programming	101
Willette Taylor	Accounting	600

Figure 7.A. Table Using "box" and Option

INPUT:

```
.TS
allbox center;
c s s
c c c
n n n.
Paradox Common Stock
Year⦁ Price⦁ Dividend
1971⦁ 41-54⦁ $2.60
2⦁ 41-54⦁ 2.70
3⦁ 46-55⦁ 2.87
4⦁ 40-53⦁ 3.24
5⦁ 45-52⦁ 3.40
6⦁ 51-59⦁ .95*
.TE
.ce
* (first quarter only)
```

OUTPUT:

Paradox Common Stock		
Year	Price	Dividend
1971	41-54	$2.60
2	41-54	2.70
3	46-55	2.87
4	40-53	3.24
5	45-52	3.40
6	51-59	.95*

* (first quarter only)

Figure 7.B. Table Using "allbox" and "center" Options

INPUT:

```
.TS
box;
c s s
c | c | c
l | l | n.
Major New York Bridges

_
Bridge⦁ Designer⦁ Length

_
Brooklyn⦁ J. A. Roebling⦁ 1595
Williamsburg⦁ L. L. Buck⦁ 1600

_
Queensborough⦁ Palmer &⦁ 1182
⦁ \0\0Hornbostel

_
⦁ ⦁ 1380
Triborough⦁ O. H. Ammann⦁ _
⦁ ⦁ 383

_
Bronx Whitestone⦁ O. H. Ammann⦁ 2300
Throgs Neck⦁ O. H. Ammann⦁ 1800
.TE
```

OUTPUT:

Major New York Bridges		
Bridge	Designer	Length
Brooklyn	J. A. Roebling	1595
Williamsburg	L. L. Buck	1600
Queensborough	Palmer & Hornbostel	1182
Triborough	O. H. Ammann	1380
		383
Bronx Whitestone	O. H. Ammann	2300
Throgs Neck	O. H. Ammann	1800

Figure 7.C. Table Using "vertical bar" Key Letter Feature

INPUT:

```
.TS
doublebox tab(:);
L L L
L L _
L L |LB
L L _
L L L.
january:february:march
april:may
june:july:Months
august:september
october:november:december
.TE
```

OUTPUT:

january	february	march
april	may	
june	july	**Months**
august	september	
october	november	december

Figure 7.D. Table Using Horizontal Lines in Place of Key Letters

INPUT:

```
.TS
box;
cfB s s s.
Composition of Foods
=
.T&
c | c s s
c | c s s
c | c | c | c.
Food⓪ Percent by Weight
\^⓪ _
\^⓪ Protein⓪ Fat⓪ Carbo-
\^⓪ \^⓪ \^⓪ hydrate
_
.T&
l | n | n | n.
Apples⓪ .4⓪ .5⓪ 13.0
Halibut⓪ 18.4⓪ 5.2⓪ ...
Lima beans⓪ 7.5⓪ .8⓪ 22.0
Mushrooms⓪ 3.5⓪ .4⓪ 6.0
Rye bread⓪ 9.0⓪ .6⓪ 52.7
.TE
```

OUTPUT:

Composition of Foods			
Food	Percent by Weight		
	Protein	Fat	Carbo-hydrate
Apples	.4	.5	13.0
Halibut	18.4	5.2	...
Lima beans	7.5	.8	22.0
Mushrooms	3.5	.4	6.0
Rye bread	9.0	.6	52.7

Figure 7.E. Table Using Additional Command Lines

INPUT:

```
.TS
allbox;
cp+1fI s s
c c cw(1.5i)
I I I.
New York Area Rocks
.sp .5
Era© Formation© Age (years)
Precambrian© Readiang Prong© >1 billion
Paleozoic© Manhattan Prong© 400 million
Mesozoic© T{
Newark Basin, incl.
Stockton,Lockatong, and Brunswick
T}© 200 million
Cenozoic© coastal Plain© T{
On Long Island 30,000 years;
Cretaceous sediments redeposited
by recent glaciation
T}
.TE
```

OUTPUT:

New York Area Rocks

Era	Formation	Age (years)
Precambrian	Reading Prong	>1 billion
Paleozoic	Manhattan Prong	400 million
Mesozoic	Newark Basin, incl. Stockton, Lockatong, and Brunswick	200 million
Cenozoic	Coastal Plain	On Long Island 30,000 years; Cretaceous sediments redeposited by recent glaciation

Figure 7.F. Table Using Text Blocks

Chapter 5: MATHEMATICS TYPESETTING PROGRAM

CONTENTS

LIST OF TABLES

Chapter 5: MATHEMATICS TYPESETTING PROGRAM

CONTENTS

LIST OF TABLES

Chapter 5: MATHEMATICS TYPESETTING PROGRAM

CONTENTS

LIST OF TABLES

Chapter 5

MATHEMATICS TYPESETTING PROGRAM

1. Introduction

Mathematical text is known in the publishing trade as "penalty copy" because it is slower, more difficult, and more expensive to set in type than any other kind of copy normally occurring in books and journals.

One difficulty is the multiplicity of characters, sizes, and fonts. Many mathematical expressions require an intimate mixture of Roman, italic, and greek letters (in three sizes) and a number of special characters. Typesetting such expressions by traditional methods is essentially a manual operation.

A second difficulty is the 2-dimensional character of mathematics. This is illustrated by the following example which shows line-drawing, built-up characters (such as braces and radicals), and a spectrum of positioning problems:

$$\int \frac{dx}{ae^{mx}-be^{-mx}} = \begin{cases} \dfrac{1}{2m\sqrt{ab}} \log \dfrac{\sqrt{a}\,e^{mx}-\sqrt{b}}{\sqrt{a}\,e^{mx}+\sqrt{b}} \\ \dfrac{1}{m\sqrt{ab}} \tanh^{-1}\left(\dfrac{\sqrt{a}}{\sqrt{b}} e^{mx}\right) \\ \dfrac{-1}{m\sqrt{ab}} \coth^{-1}\left(\dfrac{\sqrt{a}}{\sqrt{b}} e^{mx}\right) \end{cases}$$

The **eqn** software for typesetting mathematics has been designed to be easy to learn and to use by people who know neither mathematics nor typesetting. The language can be learned in an hour or so since it has few rules and fewer exceptions. It interfaces directly with the phototypesetting language, the **troff** formatter, so mathematical expressions can be embedded in the running text of a manuscript, and the entire document produced in one process. Typical mathematical expressions include size and font changes, positioning, line drawing, and other necessary functions to print according to mathematical conventions, and are done automatically. The syntax of the language is specified by a small context-free grammar; a compiler-compiler is used to make a compiler that translates this language into typesetting commands. It is

assumed that the input is to be typed on a computer terminal much like an ordinary typewriter, i.e., that **eqn** needs no special keys to input even the most complicated equation. Output may be produced on either a phototypesetter or on a terminal with forward and reverse half-line motions.

There are few rules, keywords, special symbols, and operators. This keeps the language easy to learn and remember. Furthermore, there are even fewer exceptions to the rules that do exist. If something works in one situation, it should work everywhere. If a variable can have a subscript, then a subscript can have a subscript, etc., without limit.

Subscripts and superscripts are printed automatically (with no special intervention) in appropriately smaller size. Fraction bars are made the right length and positioned at the correct height. A mechanism for overriding default actions exists, but its application is the exception, not the rule.

The system is easy to build and to change. To this end and to guarantee regularity, the language is defined by a context-free grammar. The compiler for the language was built using a compiler-compiler.

2. User Guide

Since the **eqn** program is useful for typesetting mathematics only, it interfaces with the underlying typesetting language in order to get intermingled mathematics and text. The **troff** processor performs this work for the mathematics typesetting function. **eqn** reads the input and treats as comments those things which are not mathematics passing them through untouched. (It should be noted, however, that any "comments" read by **eqn** will be converted to an *italic* font by **troff**). Text strings are passed to the **troff** formatter omitting the need for a separate storage management package. The user need not be concerned with most details of the particular device and character set currently in use. For example, the **troff** formatter computes the widths of all strings of characters; the user does not need to know about them.

At the same time, **eqn** converts mathematical inputs into **troff** formatter commands. The resulting output is passed directly to the formatter where comments and mathematical parts become text and/or formatter commands.

2.1 Delimiters

Mathematical expressions can be input by beginning and ending an equation with the **.EQ** and **.EN** delimiters, respectively. When using **troff** alone, an equation delimited with **.EQ/.EN** will be output embedded in the text surrounding it. When **troff** is used with the **−ms**, **−me** or **−mm** macro packages, the equation will be output as a display, i.e., on a line by itself preceded and followed by half a vertical space.

The **.EQ** and **.EN** delimiters are passed through to the formatter untouched, so they can be further manipulated by either pre-defined macro packages or formatter commands to, for example, center equations or number them automatically. The formatter macro packages **−me** and **−ms** allow equations to be centered, indented, left-justified by adding an argument to **.EQ**. To left-justify an equation, the **.EQ L** macro is used, and a **.EQ I** macro will indent the equation. The **−ms** and **−me** packages center equations by default. The default for **−mm** is left-justified. Any of these sequences can be followed by an arbitrary equation number placed at the right margin. For example, the input:

```
.EQ I (4.1a)
x = f ( y over 2 ) + y over 2
.EN
```

produces (using **−ms** or **−me**) the output:

$$x=f\left(\frac{y}{2}\right)+\frac{y}{2} \qquad (4.1a)$$

To get the same output when suing **−mm**, the input is:

```
.DS 1
.EQ (4.1a)
x = f ( y over 2 ) + y over 2
.EN
.DE
```

The **.EQ** and **.EN** macros can be supplemented by **troff** commands as desired. A centered equation can be produced with the input

```
.ce
.EQ
x sub i = y sub i
.EN
```

Since it is tedious to type **.EQ** and **.EN** around very short expressions (e.g., single letters), and the formatter macro packages have defined **.EQ** and **.EN** to create display equations, two characters can be defined to serve as the beginning and ending delimiters of in-line equations.

The most common character chosen for both the beginning and ending delimiter in this shorthand notation is a dollar sign ($) and is defined at the beginning of the text file by entering:

```
.EQ
delim $$
.EN
```

These characters are then recognized by **eqn** anywhere in the subsequent text. Any text between the delimiters will be treated as an equation. For example, the input:

```
This is an example of an in-line
equation $ x sub y + y = z $ using delimiters.
```

would output:

This is an example of an in-line
equation $x_y+y=z$ using delimiters.

In order to produce something like γ-ray, it is easily input using the in-line equation shorthand, for example:

```
$ gamma $-ray
```

To turn off the delimiters so that the chosen **eqn** delimiter can be used within the text without invoking **eqn**, enter the following into your file:

```
.EQ
delim off
.EN
```

Thereafter, **eqn** will no longer recognize the delimiter symbol.

Note: The following should be observed when using the in-line equations format:

- Do not use braces, tildes, circumflexes, or double quotes as delimiters as these have special significance to the **.EQ** and **.EN** macros.
- In-line font changes must be closed before in-line equations are encountered.

2.2 Spaces and New Lines

2.2.1 Input Spaces

Input is free form. Space and newline characters in the input are used by **eqn** to separate pieces of the input; they are not used to create space in the output. Thus an input such as:

```
.EQ
x   =   y
  + z + 1
.EN
```

produces the output:

$$x=y+z+1$$

Free-form input is easier to type initially. Space and newline characters should be freely used to make input equations readable and easy to edit. Very long lines are hard to correct if a mistake is made.

For the in-line equations, if impossible, the formatter will try to keep the text between the delimiters on one line. If the equation is very long **troff** will break the equation based on the spacing of characters not mathematical logic and may force the spacing between words of text to be larger than is preferable. This can be prevented by dividing the in-line equation in appropriate sections. For example:

```
$ x + y = $ $ ( c sub d ) $ $ + pi $
```

2.2.2 Output Spaces

Extra white space can be forced into the output by several characters of various sizes. A tilde (~) gives a space equal to the normal word spacing in text, a circumflex (^) gives half this much, and a tab character spaces to the next tab stop (tab stops must be set by **troff** commands). Spaces, tildes, circumflexes, and tabs also serve to delimit pieces of

input. For example, the input:

```
.EQ
x~=~y~+~z
.EN
```

produces the output:

$$x = y + z$$

2.3 Symbols, Special Names, and Greek Alphabet

Mathematical symbols, mathematical names, and the Greek alphabet are known by **eqn**. For example, the input:

```
.EQ
x=2 pi int sin ( omega t)dt
.EN
```

produces the output:

$$x=2\pi\int \sin(\omega t)\, dt$$

Spaces in the input are necessary to indicate that *sin*, *pi*, *int*, and *omega* are separate entities and should get special treatment. The **eqn** program looks up each string of characters in a table, and if found, gives it a translation. Digits, parentheses, brackets, punctuation marks, and the following mathematical words are converted to Roman font:

```
sin   cos  tan  sin  cos  tan  arc
max   min  lim  log  ln   exp
Re    Im   and  if   for  det
```

In the previous example, *pi* and *omega* become their Greek equivalents (π and ω), *int* becomes the integral sign (which is moved down and enlarged), and *sin* is output in Roman font, following conventional mathematical practice. Parentheses, digits, and operators are output in Roman font.

Spaces should be put around separate parts of the input. A common error is to type "f(pi)" without leaving spaces on both sides of the "pi". As a result, **eqn** does not recognize *pi* as a special word, and it appears as "*f(pi)*" in the output. A list of **eqn** names appears in Table 2.3 at the end of this chapter. Four-character **troff** names can also be used for anything **eqn** does not recognize, e.g., "\(**pl**" for the + sign.

The only way **eqn** can deduce that some sequence of letters may be special is if that sequence is separated from the letters on either side of it. This can be done by surrounding a special word by ordinary space, tab, or newline characters. Special words can also be made to stand out by surrounding them with tildes or circumflexes, e.g.:

```
.EQ
x~=~2~pi~int~sin~(~omega~t~)~dt
.EN
```

is much the same as the previous example, except tildes separate words like *sin*, *omega*, etc., and also add an extra space in the output per tilde. The output of this example is:

$$x = 2\ \pi \int \sin\ (\ \omega\ t\ )\ dt$$

2.4 Subscripts and Superscripts

Subscripts and superscripts are introduced by the keywords "sub" and "sup":

```
.EQ
x sup 2~+~y sub k
.EN
```

produces:

$$x^2 + y_k$$

The **eqn** program takes care of all size changes and vertical motions needed to make the hard copy look right. The words "sub" and "sup" must be surrounded by spaces. A space or tilde is used to mark the end of a subscript or superscript. Return to the original base line is automatic.

Multiple levels of subscripts or superscripts are allowed. Subscripted subscripts and superscripted superscripts such as:

```
.EQ
x sub i sub 1
.EN
```

produces:

$$x_{i_1}$$

A subscript and superscript on the same thing are printed one above

the other if the subscript comes first. The construct "something sub something sup something" is recognized as a special case:

```
.EQ
x sub i sup 2
.EN
```

outputs:

x_i^2

Other than this special case, "sub" and "sup" group to the right:

```
.EQ
x sup y sub z
.EN
```

generates:

x^{y_z}

not

x^{y}_{z}

A common erroneous expression is of the form:

```
.EQ
y = (x sup 2)+1
.EN
```

which causes

$y=(x^{2)+1}$

instead of the intended

$y=(x^2)+1$

The error is in omitting a delimiting space. The correct input expression is

```
y = ( x sup 2 ) + 1
```

2.5 Braces

Complicated expressions can be formed by using braces ({ }) to keep objects together in unambiguous groups. Braces indicate what goes over what or what terms are to be grouped before applying another

mathematical function.

Normally, the end of a subscript or superscript is marked by a space, tilde, circumflex, or tab. If the subscript or superscript is something that has to be typed with spaces in it, braces are used to mark the beginning and end. The input:

```
.EQ
e sup {i omega t}
.EN
```

produces the output:

$e^{i\omega t}$

Braces can be used to force **eqn** to treat something as a unit or just to make the intent perfectly clear.

Braces can occur within braces if necessary. The statement:

```
.EQ
e sup {i pi sup {rho +1}}
.EN
```

generates:

$e^{i\pi^{\rho+1}}$

A general rule is that an arbitrarily complicated string enclosed in braces can be used in place of a single character (such as *x*). The **eqn** program administers formatting details. In all cases, complete pairs of braces must be used. Omitting one or adding an extra one causes **eqn** to complain.

The braces convention is an example of the power of using a recursive grammar to define the language. It is part of the language that dictates if a construct can appear in some context, then any expression within braces can also occur in that context.

2.6 Fractions

Fractions are specified with the keyword *over*.

```
.EQ
a+b over c+d+e = 1
.EN
```

produces

$$\frac{a+b}{c+d+e}=1$$

The division line is made the correct length and positioned automatically. When there is both an "over" and a "sup" in the same expression, **eqn** performs the "sup" first.

```
.EQ
-b sup 2 over pi
.EN
```

is

$$\frac{-b^2}{\pi}$$

2.7 Square Roots

There is a *sqrt* operator for making square roots of the appropriate size.

```
.EQ
x = {-b +- sqrt{b sup 2 -4ac}} over 2a
.EN
```

yields

$$x=\frac{-b\pm\sqrt{b^2-4ac}}{2a}$$

Note: Since large radicals look poor on some typesetters, *sqrt* is not recommended for tall expressions.

2.8 Summations, Integrals, and Similar Constructions

Summations, integrals, and similar constructions are easy. The input:

```
.EQ
sum from i=0 to {i= inf} x sup i
.EN
```

produces:

$$\sum_{i=0}^{i=\infty} x^i$$

Braces indicate where the upper part ("i= inf") begins and ends. None are necessary for the lower part ("i=0") because it contains no spaces. Braces will never hurt; but if the "from" and "to" parts contain any spaces, braces must be put around them.

The "from" and "to" parts of the construction are optional; but if both are used, they have to occur in that order.

Other useful characters can replace the *sum* in the above example. They are:

```
int        prod        union        inter
```

which become, respectively:

$$\int \qquad \prod \qquad \cup \qquad \cap$$

Since characters before the "from" can be anything, even something in braces, "from-to" can often be used in unexpected ways. The input:

```
.EQ
lim from {n -> inf} x sub n =0
.EN
```

produces the output:

$$\lim_{n\to\infty} x_n=0$$

2.9 Size and Font Changes

Although **eqn** makes an attempt to use correct sizes and fonts, there are times when default assumptions are not what is wanted. Slides and transparencies often require larger characters than normal text. Thus size and font changing commands are also provided. By default, equations are set in 10-point type with standard mathematical conventions to determine what characters are in Roman and italic font. Size changes are made with *size n* and font changes with *roman*, *italic*, *bold*, or *fat* operations. As with the "sub" and "sup" keywords, size and font changes affect only the string that follows and revert to the normal situation afterward. Thus, the input:

```
.EQ
bold x y
.EN
```

produces:

$\mathbf{x}y$

Braces can be used if something more complicated than a single letter is to be affected. The input:

```
.EQ
bold {x y} z
.EN
```

produces:

$\mathbf{xy}z$

If fonts other than Roman, italic, and bold are to be used, the *font X* statement (*X* is a 1-character **troff** name or number for the font) can be used. Since **eqn** is tuned for Roman, italic, and bold fonts, other fonts may not give as good an appearance.

The *fat* operation takes the current font and widens it by overstriking. For instance:

```
.EQ
A = fat {pi r sup 2}
.EN
```

produces

$A=\pi r^2$

Legal sizes which may follow *size* are:

6, 7, 8, 9, 10, 11, 12, 14, 16, 18, 20, 22, 24, 28, 36.

The size can also be changed by a given amount. For example:

```
size +2
```

makes the size two points larger. This has the advantage that knowledge of the current size is not necessary.

If an entire document is to be in a nonstandard size or font, it is a nuisance to write out a size and font change for each equation. Accordingly, a global size or font can be set that thereafter affects all equations. The following statements would appear at the beginning of any equation to set the size to 16 and the font to Roman:

```
.EQ
gsize 16
gfont R
...
.EN
```

In place of "R", any of the **troff** font names may be used. The size after *gsize* can also be a relative change with + or −.

Generally, *gsize* and *gfont* appear at the beginning of a document. They can also appear throughout a document. The global font and size can be changed as often as needed, for example, in a footnote in which the size of equations should match the size of the footnote text. Footnote text is usually two points smaller than the main text. Global size should be reset at the end of the footnote.

2.10 Diacritical Marks

Diacritical marks, a problem in traditional typesetting, are straightforward in **eqn**. There are several words used to get marks on top of letters.

INPUT	OUTPUT
x dot	$\dot{x}$
x dotdot	$\ddot{x}$
x hat	$\hat{x}$
x tilde	$\tilde{x}$
x vec	$\vec{x}$
x dyad	$\overleftrightarrow{x}$
x bar	$\bar{x}$
x under	$\underline{x}$

The diacritical mark is placed at the correct height, and *bar* and *under* are made the right length for the entire construct. Other marks are centered. An example of an expression using diacritical marks is:

```
.EQ
x dot under + x hat + y tilde
+ X hat + Y dotdot = z+Z bar
.EN
```

will output:

$$\underline{\dot{x}}+\hat{x}+\tilde{y}+\hat{X}+\ddot{Y}=\overline{z+Z}$$

2.11 Quoted Text

An input entirely within quotes ("...") is not subject to font changes or spacing adjustments normally done by the typesetting program. This provides for individual spacing and adjusting if needed. For example:

```
.EQ
bold "sin (x)" + sin (x)
.EN
```

produces:

$$\textbf{sin (x)}+\sin(x)$$

Quotes are also used to get braces and other **eqn** keywords printed. The input:

```
.EQ
"{ alpha is the name for the Greek letter"~alpha "}"
.EN
```

prints:

{ *alpha is the name for the Greek letter* α}

The "" construction is often used as a place-holder when grammatically **eqn** needs something, but nothing is actually wanted on the output. For instance, **eqn** does not accept a right bracket, brace, or parenthesis without a left one. Trying to do so will produce an error message and may prevent the successful printing of the equation. However, the input:


```
.EQ
left ""
x over y
right ]
.EN
```

produces:

$$\left.\frac{x}{y}\right]$$

This point is elaborated in section 2.13.

The "" construction can also be used to add a superscript to a word in the text when the word itself will not be within the **eqn** delimiters. For example, the input:

```
What happened to the water in the mathematician's freezer?
It got ice$ "" sup 3 $ (ice cubed).
```

produces:

What happened to the water in the mathematician's freezer?
It got ice^3 (ice cubed).

2.12 Aligning Equations

Sometimes it is necessary to align a series of equations at a horizontal position, often at an equals sign. This is done with two operations called *mark* and *lineup*.

The word *mark* may appear once at any place in an equation. It remembers the horizontal position where it appeared. Successive equations can contain one occurrence of the word *lineup*. The place where *lineup* appears is made to line up with the place marked by the previous *mark* if at all possible. For example, the input:

```
.EQ
x+y mark = z
.EN
.EQ
x lineup = 1
.EN
```

produces:

$$x+y=z$$
$$x=1$$

When **eqn** and **−ms** are used, either **.EQ I** or **.EQ L** should be used. The *mark* and *lineup* operations do not work with centered equations. Also, *mark* does not look ahead:

```
.EQ
x mark = 1
.EN
.EQ
x+y lineup = z
.EN
```

is not going to work because there is not room for the "x+y" part after the *mark* remembers where the "x" is. In order to correctly line up the example equations, the following input is necessary:

```
.EQ
x = mark 1
.EN
.EQ
x + y = lineup z
.EN
```

will produce:

$$x=1$$
$$x+y=z$$

2.13 Big Brackets

To get large brackets [ ], braces { }, parentheses ( ), and bars | | around information that exists on more than one line, the *left* and *right* keywords are used:

```
.EQ
left { a over b + 1 right }
 = left ( c over d right )
+ left [ e right ]
.EN
```

produces:

$$\left\{\frac{a}{b}+1\right\}=\left(\frac{c}{d}\right)+\left[e\right]$$

The resulting brackets are made large enough to cover whatever they

enclose. Other characters can be used besides these, but they are not likely to look very good. Exceptions are the *floor* and *ceiling* characters:

```
.EQ
left floor x over y right floor
<= left ceiling a over b right ceiling
.EN
```

produces:

$$\left\lfloor\frac{x}{y}\right\rfloor\leqslant\left\lceil\frac{a}{b}\right\rceil$$

Braces are larger than brackets and parentheses because they are made up of three, five, seven, etc., pieces while brackets can be made up of two, three, four, etc., pieces. Large left and right parentheses often look strange because of the design of the character set.

The *right* keyword may be omitted. A "left something" need not have a corresponding "right something". If the right part is omitted, braces are put around the thing that the left bracket is to encompass. Otherwise, resulting brackets may be too large. If the left part is to be omitted, things are more complicated because technically a *right* cannot exist without a corresponding *left*. Instead, the following input will do:

```
.EQ
left "" ... right )
.EN
```

The *left* "" means a "left nothing" which satisfies the rules without hurting the output.

2.14 Piles

Large braces, brackets, parenthesis, and vertical bars are often used with another facility (*piles*) which makes vertical piles of objects. Elements of the pile (there can be any number) are centered one above another, at the right height for most purposes. The keyword *above* is used to separate the pieces; braces are used around the entire list. Elements of a pile can be as complicated as needed, even containing more piles.

Three other forms of pile exist:

- *lpile* makes a pile with the elements left-justified
- *rpile* makes a right-justified pile
- *cpile* makes a centered pile, just like *pile*.

Vertical spacing between pieces is somewhat larger for *lpile*, *rpile*, and *cpile* than it is for ordinary piles. For example, to get

$$sign(x) \equiv \begin{cases} 1 & \text{if } x>0 \\ 0 & \text{if } x=0 \\ -1 & \text{if } x<0 \end{cases}$$

the following is input:

```
.EQ
sign (x) == left "{"
  rpile {1 above 0 above -1}~~
  lpile {if above if above if}~~
  lpile {x>0 above x=0 above x<0}
.EN
```

The "*left* "{"" construction makes a left brace large enough to enclose the "*rpile* {...}", which is a right-justified pile. The *lpile* construction makes a left-justified pile.

2.15 Matrices

It is possible to make matrices. For example, to make a neat array like

$$x_i \quad x^2$$
$$y_i \quad y^2$$

the following is the input:

```
.EQ
matrix {
 ccol { x sub i above y sub i }
 ccol { x sup 2 above y sup 2 }
}
.EN
```

This produces a matrix with two centered columns. Elements of the columns are then listed just as for a pile. Each element is separated by the word "above". The *lcol* or *rcol* keyword can also be used to left- or right-justify columns. Each column can be separately adjusted, and there can be as many columns as desired.

The reason for using a matrix instead of two adjacent piles is if the elements of the piles are not all the same height they will not line up properly. A matrix forces them to line up because it looks at the entire structure before deciding the spacing to use.

Note: Each column must have the same number of elements. To make force each column to have the same number of elements it is possible to use the **eqn** keyword "nothing" which will give the construction the proper number of elements. For instance, the input:

```
.EQ
matrix {
  ccol { x above y sub 1 above z sup 2 }
  ccol { z above nothing above z sub 1 }
}
.EN
```

produces:

$$x \quad z$$
$$y_1$$
$$z^2 \quad z_1$$

2.16 Defines

There is a definition facility within **eqn** which allows a user to define an equation or part of an equation. Henceforth, any occurrence of *name* within an **eqn** expression will be expanded into whatever was inside the quotes in its definition. This lets users tailor the language to their own specifications. It is possible include keywords like *sup*, *sub*, or *over* in a definition. For example, if the sequence:

```
.EQ
x sub i sub 1 + y sub i sub 1
.EN
```

appears repeatedly throughout a paper; typing time can be saved each time the sequence is used by defining it:

```
.EQ
define xy 'x sub i sub 1 + y sub i sub 1'
.EN
```

This define makes *xy* a shorthand for whatever characters occur between the single quotes in the definition. (Any character can be used instead of the quote to mark the beginning and end of the definition as

long as it does not appear inside the definition.) Therefore, after defining *xy*, the input:

```
.EQ
roman "The definition xy now expands to read" ~ xy
.EN
```

produces:

The definition xy now expands to read $x_{i_1}+y_{i_1}$

Each occurrence of *xy* will expand into its definition. Spaces (or their equivalent) are to be left around the name when used. The **eqn** program will identify it as special.

Although definitions can use previous definitions, as in:

```
.EQ
define  xi  ' x sub i '
define  xi1  ' xi sub 1 '
.EN
```

it is erroneous to define something in terms of itself. For instance:

```
define  X  ' roman X '
```

Since *X* is now defined in terms of itself, problems will result. However, if the following expression is used, the quotes protect the second *X*, and everything works fine.

```
define  X  ' roman "X" '
```

The **eqn** keywords can be redefined. By making / mean *over* with the following statement:

```
.EQ
define  /  ' over '
.EN
```

or by redefining *over* as / with:

```
.EQ
define  over  ' / '
.EN
```

the keyword is redefined.

If different things are needed to be printed on a terminal and on the typesetter, symbols may be defined differently in **neqn** and **eqn**. This can be done with *ndefine* and *tdefine*. A definition made with *ndefine* takes effect when running **neqn**. When *tdefine* is used, the definition applies only for the **eqn** processor. Names defined with the *define* facility apply to both **eqn** and **neqn**.

2.17 Local Motions

Although the **eqn** formatter tries to position things correctly on the paper, it occasionally needs tuning to make the output just right. Small extra horizontal spaces can be obtained with tilde and circumflex. By using *back n* and *fwd n*, small amounts are moved horizontally, where *n* is how far to move in 1/100's of an em (an em is about the width of the letter "m"). Thus, *back 50* moves back about half the width of an "m". Similarly, things can be moved up or down with an *up n* and a *down n*. As with *sub* or *sup*, local motions affect the next thing in the input, and this can be something arbitrarily complicated if it is enclosed in braces.

2.18 Precedence

Precedence rules resolve the ambiguity in a construction like

```
.EQ
a sup 2 over b
.EN
```

The "sup" is defined to have a higher precedence than "over". A user can force a particular analysis by placing braces around expressions. If braces are not used to group functions, the **eqn** formatter will do operations in the following order:

```
dyad  vec    under  bar   tilde  hat   dot   dotdot
fwd   back   down   up
fat   roman  italic bold  size
sub   sup    sqrt   over
from  to
```

The following operations group to the left:

```
over  sqrt   left   right
```

All others group to the right.

3. Usage

On the UNIX operating system, the phototypesetter is driven by a text formatting program, **troff**, which was designed for typesetting text. Facilities needed for printing mathematical expressions, such as arbitrary horizontal and vertical motions, line drawing, and font size changing are also provided. Syntax for describing these special operations is difficult to learn and difficult even for experienced users to type correctly. For this reason, the **troff** formatter is used as an assembly language by the **eqn** program which describes and compiles mathematical expressions.

Running a preprocessor is easy on the UNIX operating system. To typeset text stored in *files*, the following command is issued:

eqn *files* | **troff**

The vertical bar connects the output of one **eqn** process to the input of another **troff** process. Any **troff** formatter options are located following the **troff** formatter part of the command. For example:

eqn filename | troff -ms

A compatible version of **eqn** can be used on devices like TELETYPE® Model 37, DASI, and GSI terminals which have half-line forward and reverse capabilities. Input language is identical, but **neqn** and the **nroff** formatter are used instead of **eqn** and the **troff** formatter. Some things will not look as good because terminals do not provide the variety of characters, sizes, and fonts that a typesetter does, but the output is usually adequate for proofreading.

To print equations on a TELETYPE® Model 37, the following command is used:

neqn *files* | **nroff**

To use a GSI or DASI terminal as the output device, the following command is used:

neqn *files* | **nroff -T***x*

where *x* is the terminal type being used, such as 300 or 300S.

The **eqn** and **neqn** programs can be used with the **tbl** program for typesetting tables that contain mathematics

tbl *files* | **eqn** | **troff**
tbl *files* | **neqn** | **nroff**

Missing delimiters and some equation errors can be detected early with program aids. Using the troubleshooting devices described in section 5 should be considered as an initial step in formatting a document.

4. Troubleshooting

If a mistake is made in an equation, such as omitting a brace, having one too many braces, or having a "sup" with nothing before it, the **eqn** formatter produces the following message:

syntax error between lines x and y, file z

where *x* and *y* are approximately the lines between which the trouble occurred, and *z* is the name of the file in question. There are also self-explanatory messages that arise when a quote is omitted or **eqn** is run on a nonexistent file. To check a document before printing

eqn *files* > /dev/null

discards the output but prints the message.

It is easy to leave out a dollar sign when used as delimiters. The **checkeq** program checks for misplaced or missing dollar signs (in-line delimiters) and similar troubles.

In-line equations can be only so big because of an internal buffer in the **troff** formatter. If a "word overflow" message is received, the limit has been exceeded. One solution is to break the equation into smaller units with the in-line delimiters. Also, printing the equation as a displayed equation usually causes the message to go away. The "line overflow" message indicates that an even bigger buffer has been exceeded. In this case, the equation must be broken into two separate ones, marking each with **.EQ/.EN** delimiters. The **eqn** program does not warn about equations that are too long for one line.

TABLE 2.3. Naming Convention for the EQN

CHARACTER SEQUENCE	OUTPUT
> =	≥
< =	≤
= =	≡
! =	≠
+ −	±
− >	→
< −	←
< <	≪
> >	≫
inf	∞
partial	∂
half	½
prime	′
approx	≈
nothing	
cdot	·
times	×
del	∇
grad	∇
...	...
,....,	, ... ,
sum	Σ
int	∫
prod	Π
union	∪
inter	∩

INPUT NAME	CHARACTER
DELTA	Δ
GAMMA	Γ
LAMBDA	Λ
OMEGA	Ω
PHI	Φ
PI	Π
PSI	Ψ
SIGMA	Σ
THETA	Θ
UPSILON	Υ
XI	Ξ
alpha	α
beta	β
chi	χ
delta	δ
epsilon	ϵ
eta	η
gamma	γ
iota	ι
kappa	κ
lambda	λ
mu	μ
nu	ν
omega	ω
omicron	o
phi	ϕ
pi	π
psi	ψ
rho	ρ
sigma	σ
tau	τ
theta	θ
upsilon	υ
xi	ξ
zeta	ζ

Chapter 6: MEMORANDUM MACROS

CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 6

MEMORANDUM MACROS

1. Introduction

1.1 Purpose

This chapter is a guide and reference for users of the Memorandum Macros. These macros provide a general purpose package of text formatting macros for use with the UNIX operating system text formatters **nroff** and **troff** (refer to **nroff** and **troff**(1) in the *UniPlus*$^+$ *User's Manual*, Section 1, for more details).

Note: A reference of the form **name**(*N*) points to page **name** in section *N* of the *UniPlus*$^+$ *User's Manual.*

1.2 Conventions

Each part of this section explains a single facility of **mm** and progresses from general case to special-case facilities. It is recommended that a user read a part in detail only to the point where there is enough information to obtain the desired format, then skim the rest because some details may be of use to only a few.

Numbers enclosed in brackets ({ }) refer to section numbers within this chapter. For example, this is section {1.2}.

In the synopses of macro calls, square brackets ([ ]) surrounding an argument indicate that it is optional. Ellipses (...) show that the preceding argument may appear more than once.

Figure 16.A at the end of this chapter shows both **nroff** and **troff** formatter outputs (of files using **mm** macros) for a simple letter.

1.3 Document Structure

Input for a document to be formatted with the **mm** text formatting macro package has four major segments, any of which may be omitted; if present, the segments must occur in the following order:

- *Parameter setting segment* sets the general style and appearance of a document. The user can control page width, margin

justification, numbering styles for heading and lists, page headers and footers, and many other properties of the document. Also, the user can add macros or redefine existing ones. This segment can be omitted entirely if the user is satisfied with default values; it produces no actual output, but performs only the formatter setup for the rest of the document.

- *Beginning segment* includes those items that occur only once, at the beginning of a document, e.g., title, author's name, date.
- *Body segment* is the actual text of the document. It may be as small as a single paragraph or as large as hundreds of pages. It may have a hierarchy of headings up to seven levels deep {4}. Headings are automatically numbered (if desired) and can be saved to generate the table of contents. Five additional levels of subordination are provided by a set of list macros for automatic numbering, alphabetic sequencing, and "marking" of list items {5}. The body may also contain various types of displays, tables, figures, footnotes, and references {7, 8, 11}.
- *Ending segment* contains those items that occur only once at the end of a document. Included are signature(s) and lists of notations (e.g., "Copy to" lists) {6.11}. Certain macros may be invoked here to print information that is wholly or partially derived from the rest of the document, such as the table of contents or the cover sheet for a document {10}.

Existence and size of these four segments varies widely among different document types. Although a specific item (such as date, title, author names, etc.) of a segment may differ depending on the document, there is a uniform way of typing it into an input text file.

1.4 Input Text Structure

In order to make it easy to edit or revise input file text at a later time:

- Input lines should be kept short.
- Lines should be broken at the end of clauses.
- Each new sentence should begin on a new line.

1.5 Definitions

Formatter refers to either the **nroff** or **troff** text-formatting program.

Requests are built-in commands recognized by the formatters. Although a user seldom needs to use these requests directly {3.10}, this chapter contains references to some of the requests. For example, the request

```
.sp
```

inserts a blank line in the output at the place the request occurs in the input text file.

Macros are named collections of requests. Each macro is an abbreviation for a collection of requests that would otherwise require repetition. The **mm** package supplies many macros, and the user can define additional ones. Macros and requests share the same set of names and are used in the same way. Table 16.A at the end of this chapter is an alphabetical list of macro names used by **mm**. The first line of each item lists the name of the macro, a brief description, and a reference to the section in which the macro is described. The second line illustrates a typical macro structure.

Strings provide character variables, each of which names a string of characters. Strings are often used in page headers, page footers, and lists. These registers share the pool of names used by requests and macros. A string can be given a value via the **.ds** (define string) request, and its value can be obtained by referencing its name, preceded by "*" (for 1-character names) or "*(" (for 2-character names). For instance, the string **DT** in **mm** normally contains the current date, thus the input line

```
Today is \*(DT.
```

may result in the following output:

```
Today is September 15, 1984.
```

The current date can be replaced, e.g.:

```
.ds DT 01/01/85
```

by invoking a macro designed for that purpose {6.8}. Table 16.B at the end of this chapter is an alphabetical list of string names used by **mm**. A brief description, paragraph reference, and initial (default) value(s) are given for each.

Number registers fill the role of integer variables. These registers are used for flags and for arithmetic and automatic numbering. A register can be given a value using a **.nr** request and be referenced by preceding its name by **\n** (for 1-character names) or **\n(** (for 2-character names). For example, the following sets the value of the register **d** to one more than that of the register **dd**:

```
.nr d 1+\n(dd
```

Table 16.C at the end of this chapter is an alphabetical list of number register names giving for each a brief description, paragraph reference, initial (default) value, and legal range of values.

Section 14.1 contains naming conventions for requests, macros, strings, and number registers. Tables 16.A, 16.B, and 16.C lists all macros, strings, and number registers defined in **mm**.

2. Usage

This part describes how to access **mm**, illustrates UNIX operating system command lines appropriate for various output devices, and describes command line flags for the **mm** text-formatting macro package.

2.1 The mm Command

The **mm**(1) command can be used to prepare documents using the **nroff** formatter and the Memorandum Macros; this command invokes **nroff** with the **−cm** flag {2.2}. The **mm** command has options to specify preprocessing by **tbl** and/or by **neqn** and for postprocessing by various output filters.

Note: Options can occur in any order but must appear before the file names.

Any arguments or flags that are not recognized by the **mm** command, e.g., **−rC3**, are passed to the **nroff** formatter or to **mm**, as appropriate. Options are:

OPTION	MEANING
−e	The **neqn** is to be invoked; also causes **neqn** to read **/usr/pub/eqnchar** [see **eqnchar**(7)].

−t	The **tbl**(1) processor is to be invoked.
−c	The **col**(1) postprocessor is to be invoked.
−E	The **−e** option of the **nroff** formatter is to be invoked.
−y	The **−mm** (uncompacted macros) is to be used instead of **−cm**.
−12	The 12-pitch mode is to be used. The pitch switch on the terminal should be set to 12 if necessary.
−T450	Output is to a DASI 450. This is the default terminal type [unless **$TERM** is set; see **sh**(1)]. It is also equivalent to **−T1620**.
−T450−12	Output is to a DASI 450 in 12-pitch mode.
−T300	Output is to a DASI 300 terminal.
−T300−12	Output is to a DASI 300 in 12-pitch mode.
−T300s	Output is to a DASI 300S.
−T300s−12	Output is to a DASI 300S in 12-pitch mode.
−T4014	Output is to a Tektronix 4014.
−T37	Output is to a TELETYPE® Model 37.
−T382	Output is to a DTC-382.
−T4000a	Output is to a Trendata 4000A.
−TX	Output is prepared for an EBCDIC line printer.
−Thp	Output is to a HP264x (implies **−c**).
−T43	Output is to a TELETYPE® Model 43 (implies **−c**).
−T40/4	Output is to a TELETYPE® Model 40/4 (implies **−c**).
−T745	Output is to a Texas Instrument 700 series terminal (implies **−c**).
−T2631	Output is prepared for a HP2631 printer where **−T2631−e** and **−T2631−c** may be used for expanded and compressed modes, respectively

(implies **−c**).

−Tlp Output is to a device with no reverse or partial line motions or other special features (implies **−c**).

Any other **−T** option given does not produce an error; it is equivalent to **−Tlp**.

A similar command is available for use with the **troff** formatter [see **mmt**(1)].

2.2 The −cm or −mm Flag

The **mm** package can also be invoked by including the **−cm** or **−mm** flag as an argument to the formatter. The **−cm** flag causes the precompacted version of the macros to be loaded. The **−mm** flag causes the file **/usr/lib/tmac/tmac.m** to be read and processed before any other files. This action:

- defines the Memorandum Macros,
- sets default values for various parameters, and
- initializes the formatter to be ready to process input text files.

2.3 Typical Command Lines

The prototype command lines are as follows (various options are explained in section 2.4):

- Text without tables or equations:

  mm [*options*] *filename* ...

  or

  nroff [*options*] **−cm** *filename* ...

  mmt [*options*] *filename* ...

  or

  troff [*options*] **−cm** *filename* ...

- Text with tables:

  mm −t [*options*] *filename* ...

  or

  tbl *filename* ... | **nroff** [*options*] **−cm**

  mmt −t [*options*] *filename* ...

  or

  tbl *filename* ... | **troff** [*options*] **−cm**

- Text with equations:

  mm −e [*options*] *filename* ...

  or

  neqn /usr/pub/eqnchar *filename* ... | **nroff** [*options*] **−cm**

  mmt −e [*options*] *filename* ...

  or

  eqn /usr/pub/eqnchar *filename* ... | **troff** [*options*] **−cm**

- Text with both tables and equations:

  mm −t −e [*options*] *filename* ...

  or

  tbl *filename* ... | **neqn /usr/pub/eqnchar** | **nroff** [*options*] **−cm**

  mmt −t −e [*options*] *filename* ...

  or

  tbl *filename* ... | **eqn /usr/pub/eqnchar** | **troff** [*options*] **−cm**

When formatting a document with the **nroff** processor, the output should normally be processed for a specific type of terminal because the output may require some features that are specific to a given terminal (e.g., reverse paper motion or half-line paper motion in both directions). Some commonly used terminal types and the command lines appropriate for them are given below. More information is found in section 2.4 of this chapter and in **300**(1), **450**(1), **4014**(1), **hp**(1), **col**(1), **termio**(4), and **term**(5) of the *UniPlus*$^{+}$ *User's Manual.*

- DASI 450 in 10-pitch, 6 lines/inch mode, with 0.75 inch offset, and a line length of 6 inches (60 characters) where this is the default terminal type so no **−T** option is needed (unless **$TERM** is set to another value):

mm *filename* ...
or
nroff −T450 −h −cm *filename* ...

- DASI 450 in 12-pitch, 6 lines/inch mode, with 0.75 inch offset, and a line length of 6 inches (72 characters):

  mm −12 *filename* ...
  or
  nroff −T450-12 −h −cm *filename* ...

  or to increase the line length to 80 characters and decrease the offset to 3 characters:

  mm −12 −rW80 −rO3 *filename* ...
  or
  nroff −T450-12 −rW80 −rO3 −h −cm *filename* ...

- Hewlett-Packard HP264x CRT family:

  mm −Thp *filename* ...
  or
  nroff −cm *filename* ... | **col** | **hp**

- Any terminal incapable of reverse paper motion and also lacking hardware tab stops (Texas Instruments 700 series, etc.):

  mm −T745 *filename* ...
  or
  nroff −cm *filename* ... | **col −x**

The **tbl**(1) and **eqn/neqn**(1) formatters must be invoked as shown in the command lines illustrated earlier.

If 2-column processing {12.4} is used with the **nroff** formatter, either the **−c** option must be specified to **mm**(1) [**mm**(1) uses **col**(1) automatically for many terminal types {2.1}] or the **nroff** formatter output must be postprocessed by **col**(1). In the latter case, the **−T37** terminal type must be specified to the **nroff** formatter, the **−h** option must not be specified, and the output of **col**(1) must be processed by the appropriate terminal filter [e.g., **450**(1)]; **mm**(1) with the **−c** option handles all this automatically.

2.4 Parameters Set From Command Line

Number registers are commonly used within **mm** to hold parameter values that control various aspects of output style. Many of these values can be changed within the text files with **.nr** requests. In addition, some of these registers can be set from the command line. This is a useful feature for those parameters that should not be permanently embedded within the input text. If used, the number registers (with the exception of the **P** register) must be set on the command line or before the **mm** macro definitions are processed. The number register meanings are:

−rA*n* — *n* = 1, has effect of invoking the **.AF** macro without an argument {6.9}.
n = 2, permits use of Bell System logo, if available, on a printing device (currently available for Xerox 9700 only).

−rC*n* — sets type of copy (e.g., DRAFT) to be printed at bottom of each page {9.2.4}.
n = 1, OFFICIAL FILE COPY.
n = 2, DATE FILE COPY.
n = 3, DRAFT with single spacing and default paragraph style.
n = 4, DRAFT with double spacing and 10-space paragraph indent.

−rD1 — sets *debug* mode.
This flag requests formatter to continue processing even if **mm** detects errors that would otherwise cause termination. It also includes some debugging information in the default page header {9.2.1, 12.3}.

−rE*n* — controls font of Subject/Date/From fields.
n = 0, fields are bold (default for the **troff** formatter).
n = 1, fields are Roman font (regular text default for the **nroff** formatter).

−rL*k* — sets length of physical page to *k* lines.
For the **nroff** formatter, *k* is an unscaled number representing lines.
For the **troff** formatter, *k* must be scaled (i.e., **i** for inches, **v** for vertical spaces).
Default value is 66 lines per page.
This flag was used, for example, when printing the 6-by-9 format of this document (i.e., **−rL9i**).

−rN*n* specifies page numbering style.
n = 0, (default) all pages get the prevailing header {9.2.1}.
n = 1, page header replaces footer on page 1 only.
n = 2, page header is omitted from page 1.
n = 3, "section-page" numbering {4.5} occurs (**.FD** {8.3} and **.RP** {11.4} define footnote and reference numbering in sections).
n = 4, default page header is suppressed; however, a user-specified header is not affected.
n = 5, "section-page" and "section-figure" numbering occurs.

n	PAGE 1	PAGES 2ff
0	header	header
1	header replaces footer	header
2	no header	header
3	"section-page" as footer	same as page 1
4	no header	no header unless **.PH** defined
5	"section-page" as footer and "section-figure"	same as page 1

Contents of the prevailing header and footer do not depend on number register *N* value; *N* controls only whether the header (*N*=3) or the footer (*N*=5) is printed, as well as the page numbering style. If header and footer are null {9.2.1, 9.2.4}, the value of *N* is irrelevant.

−rO*k* offsets output *k* spaces to the right.
For the **nroff** formatter, *k* is an unscaled number representing character positions.
For the **troff** formatter, *k* must be scaled.
This flag is helpful for adjusting output positioning on some terminals. If this register is not set on the command line, the default offset is 0.75 inch.

Note: Register name is the capital letter "O".

−rP*n* specifies that pages of the document are to be numbered starting with *n*.
This register may also be set via a **.nr** request in the input text.

−rS*n* sets point size and vertical spacing for the document.
The default *n* is 10, i.e., 10-point type on 12-point vertical spacing, giving six lines per inch {12.9}.
This flag applies to the **troff** formatter only.

−rT*n* provides register settings for certain devices.
n = 1, line length and page offset are set to 80 and 3, respectively.
n = 2, changes the page length to 84 lines per page and inhibits underlining; it is meant for output sent to the Versatec® printer.
The default value for *n* is 0.
This flag applies to the **nroff** formatter only.

−rU1 controls underlining of section headings.
This flag causes only letters and digits to be underlined. Otherwise, all characters (including spaces) are underlined {4.2.2.4.2}.
This flag applies to the **nroff** formatter only.

−rW*k* sets page width (line length and title length) to *k*.
For the **nroff** formatter, *k* is an unscaled number representing character positions.
For the **troff** formatter, *k* must be scaled.
This flag can be used to change page width from the default value of 6 inches (60 characters in 10 pitch or 72 characters in 12 pitch).

2.5 Omission of −cm or −mm Flag

If a large number of arguments is required on the command line, it may be convenient to set up the first (or only) input file of a document as follows:

```
zero or more initializations of registers listed in section 2.4
.so /usr/lib/tmac/tmac.m
remainder of text
```

In this case, the user must not use the **−cm** or **−mm** flag [nor the **mm**(1) or **mmt**(1) command]; the **.so** request has the equivalent effect, but registers shown in section 2.4 must be initialized before the **.so** request because their values are meaningful only if set before macro definitions are processed. When using this method, it is best to lock

into the input file only those parameters that are seldom changed. For example:

```
.nr W 80
.nr O 10
.nr N 3
.so /usr/lib/tmac/tmac.m
.H 1 "INTRODUCTION"
  .
  .
  .
```

specifies, for the **nroff** formatter, a line length (W) of 80, a page offset (O) of 10, and "section-page" (N) numbering.

3. Formatting Concepts

3.1 Basic Terms

Normal action of the formatters is to fill output lines from one or more input lines. Output lines may be justified so that both the left and right margins are aligned. As lines are being filled, words may also be hyphenated {3.4} as necessary. It is possible to turn any of these modes on and off (**.SA** {12.2}, **Hy** {3.4}, and the **.nf** and **.fi** formatter requests). Turning off fill mode also turns off justification and hyphenation.

Certain formatting commands (requests and macros) cause filling of the current output line to cease, the line (of whatever length) to be printed, and subsequent text to begin a new output line. This printing of a partially filled output line is known as a *break*. A few formatter requests and most of the **mm** macros cause a break.

Formatter requests {3.10} can be used with **mm**; however, there are consequences and side effects that each such request might have. A good rule is to use formatter requests only when absolutely necessary. The **mm** macros described herein should be used in most cases because:

- It is much easier to control (and change at any later point in time) overall style of the document.
- Complicated features (such as footnotes or tables of contents) can be obtained with ease.

- User is insulated from complexities of the formatter language.

3.2 Arguments and Double Quotes

For any macro call, a null argument is an argument whose width is zero. Such an argument often has a special meaning; the preferred form for a null argument is "". Omitting an argument is not the same as supplying a null argument (e.g., the **.MT** macro {6.7}). Omitted arguments can occur only at the end of an argument list; null arguments can occur anywhere in the list.

Any macro argument containing ordinary (paddable) spaces must be enclosed in double quotes. A double quote (") is a single character that must not be confused with two apostrophes ('), acute accents (´), or grave accents (`). Otherwise, it will be treated as several separate arguments.

Double quotes are not permitted as part of the value of a macro argument or of a string that is to be used as a macro argument. If it is necessary to have a macro argument value, two grave accents (``) and/or two acute accents (´´) may be used instead. This restriction is necessary because many macro arguments are processed (interpreted) a variable number of times. For example, headings are first printed in the text and may be reprinted in the table of contents.

3.3 Unpaddable Spaces

When output lines are justified to give an even right margin, existing spaces in a line may have additional spaces appended to them. This may distort the desired alignment of text. To avoid this distortion, it is necessary to specify a space that cannot be expanded during justification, i.e., an *unpaddable space*. There are several ways to accomplish this:

- The user may type a backslash followed by a space (\ ). This pair of characters directly generates an unpaddable space.
- The user may sacrifice some seldom-used character to be translated into a space upon output.

Because this translation occurs after justification, the chosen character may be used anywhere an unpaddable space is desired. The tilde (~) is often used with the translation macro for this purpose. To use the tilde in this way, the following is inserted at the beginning of the document:

```
.tr ~
```

If a tilde must actually appear in the output, it can be temporarily "recovered" by inserting

```
.tr ~~
```

before the place where needed. Its previous usage is restored by repeating the **.tr ~** after a break or after the line containing the tilde has been forced out.

Note: Use of the tilde in this fashion is not recommended for documents in which the tilde is used within equations.

3.4 Hyphenation

Formatters do not perform hyphenation unless requested. Hyphenation can be turned on in the body of the text by specifying

```
.nr Hy 1
```

once at the beginning of the document input file. Section 8.3 describes hyphenation within footnotes and across pages.

If hyphenation is requested, formatters will automatically hyphenate words if need be. However, the user may specify hyphenation points for a specific occurrence of any word with a special character known as a hyphenation indicator or may specify hyphenation points for a small list of words (about 128 characters).

If the *hyphenation indicator* (initially, the 2-character sequence "\%") appears at the beginning of a word, the word is not hyphenated. Alternatively, it can be used to indicate legal hyphenation points inside a word. All occurrences of the hyphenation indicator disappear on output.

The user may specify a different hyphenation indicator.

.HC [*hyphenation-indicator*]

The circumflex (^) is often used for this purpose by inserting the following at the beginning of a document input text file:


```
.HC ^
```

Note: Any word containing hyphens or dashes (also known as *em dashes*) will be hyphenated immediately after a hyphen or dash if it is necessary to hyphenate the word, even if the formatter hyphenation function is turned off.

The user may supply, via the exception word **.hw** request, a small list of words with the proper hyphenation points indicated. For example, to indicate the proper hyphenation of the word "printout", the user may specify

```
.hw print-out
```

3.5 Tabs

Macros **.MT** {6.7}, **.TC** {10.1}, and **.CS** {10.2} use the formatter **.ta** (tab) request to set tab stops and then restore the default values of tab settings (every eight characters in the **nroff** formatter; every ½ inch in the **troff** formatter). Setting tabs to other than the default values is the user's responsibility.

Default tab setting values for **nroff** are 9, 17, 25, ..., 161 for a total of 20 tab stops. Values may be separated by commas, spaces, or any other non-numeric character. A user may set tab stops at any value desired. For example:

```
.ta 1.5i 3i 4.5i
```

A tab character is interpreted with respect to its position on the input line rather than its position on the output line. In general, tab characters should appear only on lines processed in no-fill (**.nf**) mode {3.1}.

The **tbl**(1) program {7.3} changes tab stops but does not restore default tab settings.

3.6 BEL Character

The nonprinting character BEL is used as a delimiter in many macros to compute the width of an argument or to delimit arbitrary text, e.g., in page headers and footers {9}, headings {4}, and lists {5}. Users who include BEL characters in their input text file (especially in arguments to macros) will receive mangled output.

3.7 Bullets

A bullet (•) is often obtained on a typewriter terminal by using an "o" overstruck by a "+". For compatibility with the **troff** formatter, a bullet string is provided by **mm** with the following sequence:

```
\*(BU
```

The bullet list (**.BL**) macro {5.4} uses this string to generate automatically the bullets for bullet-listed items.

3.8 Dashes, Minus Signs, and Hyphens

The **troff** formatter has distinct graphics for a dash, a minus sign, and a hyphen; the **nroff** formatter does not.

- Users who intend to use the **nroff** formatter only may use the minus sign (−) for the minus, hyphen, and dash.
- Users who plan to use the **troff** formatter primarily should follow **troff** escape conventions (i.e., **\(mi** for minus, **\(em** for dash and **\(hy** for hyphen).
- Users who plan to use both formatters must take care during input text file preparation. Unfortunately, these graphic characters cannot be represented in a way that is both compatible and convenient for both formatters.

The following approach is suggested:

Dash	Type "***(EM**" for each text dash for both **nroff** and **troff** formatters. This string generates an em (—) dash in the **troff** formatter and two dashes (--) in the **nroff** formatter. Dash list (**.DL**) macros {5.5} automatically generate the em dash for each list item.
Hyphen	Type "-" and use as is for both formatters. The **nroff** formatter will print it as is. The **troff** formatter will print a true hyphen.
Minus	Type "\-" for a true minus sign regardless of formatter. The **nroff** formatter will ignore the "\". The **troff** formatter will print a true minus sign (−).

3.9 Trademark String

A trademark string "*(Tm" is available with **mm**. This places the letters "TM" one-half line above the text that it follows. For example:

```
The
UniPlus+\*(Tm manual
is available from the library.
```

yields:

The UniPlus+™ manual is available from the library.

3.10 Use of Formatter Requests

Most formatter requests should not be used with **mm** because **mm** provides the corresponding formatting functions in a much more user-oriented and surprise-free fashion than do the basic formatter requests. However, some formatter requests are useful with **mm**, namely the following:

.af	assign format
.br	break
.ce	center
.de	define macro
.ds	define string
.fi	fill output lines
.hw	hyphen word exceptions
.ls	line spacing
.nf	no filling of output lines
.nr	number register define and set
.nx	next file (does not return)
.rm	remove macro
.rr	remove register
.rs	restore spacing
.so	source file and return
.sp	space
.ta	tab stop settings
.ti	temporary indent
.tl	title
.tr	translate
.!	escape

The **.fp** (font position), **.lg** (ligature mode), and **.ss** (space-character size) requests are also sometimes useful for the **troff** formatter. Use of other requests without fully understanding their implications very often leads to disaster.

4. Paragraphs and Headings

4.1 Paragraphs

.P [*type*]
one or more lines of text.

The **.P** macro is used to control paragraph style.

4.1.1 Paragraph Indention

An indented or a nonindented paragraph is defined with the *type* argument:

type	RESULT
0	left justified
1	indent

In a left-justified paragraph, the first line begins at the left margin. In an indented paragraph, the paragraph is indented the amount specified in the **Pi** register (default value is 5 ens) For example, to indent paragraphs by ten spaces in **nroff** the following is entered at the beginning of the document input file:

```
.nr Pi 10
```

A document input file possesses a default paragraph type obtained by specifying **.P** before each paragraph that does not follow a heading {4.2}. Default paragraph type is controlled by the **Pt** number register.

- The initial value of **Pt** is 0, which provides left-justified paragraphs.
- All paragraphs can be forced to be indented by inserting the following at the beginning of the document input file:

  ```
  .nr Pt 1
  ```

- All paragraphs can be indented except after headings, lists, and displays by entering the following at the beginning of the

document input file:

```
.nr Pt 2
```

Both the **Pi** and **Pt** register values must be greater than zero for any paragraphs to be indented.

Note: Values that specify indentation must be unscaled and are treated as character positions, i.e., as a number of ens. In the **nroff** formatter, an en is equal to the width of a character. In the **troff** formatter, an en is the number of points (1 point = 1/72 of an inch) equal to half the current point size.

Regardless of the value of **Pt**, an individual paragraph can be forced to be left-justified or indented. The **.P 0** macro request forces left justification; **.P 1** causes indentation by the amount specified by the register **Pi**.

If **.P** occurs inside a list, the indent (if any) of the paragraph is added to the current list indent {5}.

4.1.2 Numbered Paragraphs

Numbered paragraphs may be produced by setting the **Np** register to 1. This produces paragraphs numbered within first level headings, e.g., 1.01, 1.02, 1.03, 2.01, etc.

A different style of numbered paragraphs is obtained by using the **.nP** macro rather than the **.P** macro for paragraphs. This produces paragraphs that are numbered within second level headings.

```
.H 1 "FIRST HEADING"
.H 2 "Second Heading"
.nP
one or more lines of text
```

The paragraphs contain a "double-line indent" in which the text of the second line is indented to be aligned with the text of the first line so that the number stands out.

4.1.3 Spacing Between Paragraphs

The **Ps** number register controls the amount of spacing between paragraphs. By default, **Ps** is set to 1, yielding one blank space in **nroff**, one-half a vertical space in **troff**.

4.2 Numbered Headings

.H *level* [*heading-text*] [*heading-suffix*]
zero or more lines of text

The *level* argument provides the numbered heading level. There are seven heading levels; level 1 is the highest, level 7 is the lowest.

The *heading-text* argument is the text of the heading. If the heading contains more than one word or contains spaces, the entire argument must be enclosed in double quotes.

The *heading-suffix* argument may be used for footnote marks which should not appear with heading text in the table of contents.

There is no need for a **.P** macro immediately after a **.H** or **.HU** {4.3} because the **.H** macro also performs the function of the **.P** macro. Any immediately following **.P** macro is ignored. It is, however, good practice to start every paragraph with a **.P** macro, thereby ensuring that all paragraphs uniformly begin with a **.P** throughout an entire document.

4.2.1 Default Headings

The effect of the **.H** macro varies according to the *level* argument. First-level headings are preceded by two blank lines in **nroff** and one vertical space in **troff**; all other levels are preceded by one blank line in **nroff** and one-half a vertical space in **troff**. The following describes the default effect of the *level* argument.

.H 1 heading-text	Produces a bold font heading, one point size smaller than the text (**troff** only), followed by a single blank line (**nroff**) or one-half a vertical space (**troff**). The text that follows begins on a new line and is indented according to the current paragraph type. Full capital letters can be used to make the heading stand out.
.H 2 heading-text	Produces a bold font heading followed by a single blank line (**nroff**) or one-half a vertical space (**troff**). The text that follows begins on a new line and is indented according to the current paragraph type. Initial capitals can be used in the heading text.
.H *n* heading-text	(*n* = 3, 4, 5, 6, or 7) Produces an underlined (**nroff**) or italicized (**troff**) heading followed by two spaces The text that follows begins on the same line, i.e., these are *run-in* headings.

Appropriate numbering and spacing (horizontal and vertical) occur even if the *heading-text* argument is omitted from a **.H** macro call.

Note: Users satisfied with the default appearance of headings may skip to the section entitled "Unnumbered Headings" {4.3}.

4.2.2 Altering Appearance

The user can modify the appearance of headings quite easily by setting certain registers and strings at the beginning of the document input text file. This permits quick alteration of a document's style because this style-control information is concentrated in a few lines rather than being distributed throughout the document.

4.2.2.1 Prespacing and Page Ejection

A first-level heading (**.H 1**) normally has two blank lines (**nroff**) or one vertical space (**troff**) preceding it, and all other headings are preceded by one blank line (**nroff**) or one-half a vertical space (**troff**). If a multi-line heading were to be split across pages, it is automatically moved to the top of the next page. Every first-level heading may be forced to the top of a new page by inserting:

```
.nr Ej 1
```

at the beginning of the document input text file. Long documents may be made more manageable if each section starts on a new page. Setting the **Ej** (eject) register to a higher value causes the same effect for headings up to that level, i.e., a page eject occurs if the heading level is less than or equal to the **Ej** value.

4.2.2.2 Spacing After Headings

Three registers control the appearance of text immediately following a .H call. The registers are **Hb** (heading break level), **Hs** (heading space level), and **Hi** (post-heading indent).

If the heading level is less than or equal to the value of **Hb**, a break {3.1} occurs after the heading.

If the heading level is less than or equal to the value of **Hs**, a blank line (**nroff**) or one-half a vertical space (**troff**) is inserted after the heading.

If a heading level is greater than the value of **Hb** and also greater than the value of **Hs**, then the heading (if any) is run into the following text. These registers permit headings to be separated from the text in a consistent way throughout a document while allowing easy alteration of white space and heading emphasis. The default value for **Hb** and **Hs** is 2.

For any stand-alone heading, i.e., a heading not run into the following text, alignment of the next line of output is controlled by the **Hi** number register.

- If **Hi** is 0, text is left-justified.
- If **Hi** is 1 (the default value), text is indented according to the paragraph type as specified by the **Pt** register {4.1.1}.
- If **Hi** is 2, text is indented to line up with the first word of the heading itself so that the heading number stands out more clearly.

To cause a blank line (**nroff**) or one-half a vertical space (**troff**) to appear after the first three heading levels, to have no run-in headings, and to force the text following all headings to be left-justified (regardless of the value of **Pt**), the following should appear at the beginning of the document input text file:

```
.nr Hs 3
.nr Hb 7
.nr Hi 0
```


4.2.2.3 Centered Headings

The **Hc** register can be used to obtain centered headings. A heading is centered if its *level* argument is less than or equal to **Hc** and if it is also a stand-alone heading. The **Hc** register is 0 initially (no centered headings).

4.2.2.4 Bold, Italic, and Underlined Headings

4.2.2.4.1 Control by Level:

Any heading that is underlined by the **nroff** formatter is italicized by the **troff** formatter. The string **HF** (heading font) contains seven codes that specify fonts for heading levels 1 through 7. Legal codes, code interpretations, and defaults for **HF** codes are:

FORMATTER	HF CODE			DEFAULT HF CODE
	1	2	3	
nroff	Roman	underline	bold	3 3 2 2 2 2 2
troff	Roman	italic	bold	3 3 2 2 2 2 2

Thus, levels 1 and 2 are bold; levels 3 through 7 are underlined by the **nroff** formatter and italicized by the **troff** formatter. The user may reset **HF** as desired. Any value omitted from the right end of the list is assumed to be a 1. The following request would result in levels 1 through 5 in bold font and levels 6 and 7 in Roman font:

```
.ds HF 3 3 3 3 3
```

4.2.2.4.2 NROFF Underlining Style:

The **nroff** formatter underlines in either of two styles:

- The normal style (**.ul** request) is to underline only letters and digits.
- The continuous style (**.cu** request) underlines all characters including spaces.

By default, **mm** attempts to use the continuous style on any heading that is to be underlined and is short enough to fit on a single line. If a heading is to be underlined but is longer than a single line, the heading is underlined in the normal style (only letters and digits).

All underlining of headings can be forced to the normal style by using the **−rU1** flag when invoking the **nroff** formatter {2.4}.

4.2.2.4.3 Heading Point Sizes:

The user may specify the desired point size for each heading level with the **HP** string (for use with the **troff** formatter only).

.ds HP [*ps1*] [*ps2*] [*ps3*] [*ps4*] [*ps5*] [*ps6*] [*ps7*]

By default, the text of headings (**.H** and **.HU**) is printed in the same point size as the body except that bold stand-alone headings are printed in a size one point smaller than the body. The string **HP**, similar to the string **HF**, can be specified to contain up to seven values, corresponding to the seven levels of headings. For example:

.ds HP 12 12 10 10 10 10 10

specifies that the first- and second-level headings are to be printed in 12-point type with the remainder printed in 10-point. Specified values may also be relative point-size changes, for example:

.ds HP +2 +2 −1 −1

If absolute point sizes are specified, then absolute sizes will be used regardless of the point size of the body of the document. If relative point sizes are specified, then point sizes for headings will be relative to the point size of the body even if the latter is changed.

Null or zero values imply that default size will be used for the corresponding heading level.

Note: Only the point size of the headings is affected. Specifying a large point size without providing increased vertical spacing (via **.HX** and/or **.HZ**) may cause overprinting.

4.2.2.5 Marking Styles — Numerals and Concatenation

.HM [*arg1*] ... [*arg7*]

The registers named **H1** through **H7** are used as counters for the seven levels of headings. Register values are normally printed using Arabic numerals. The **.HM** macro (heading mark style) allows this choice to be overridden thus providing "outline" and other document styles.

This macro can have up to seven arguments; each argument is a string indicating the type of marking to be used. Legal arguments and their meanings are:

ARGUMENT	MEANING
1	Arabic (default for all levels)
0001	Arabic with enough leading zeroes to get the specified number of digits
A	Uppercase alphabetic
a	Lowercase alphabetic
I	Uppercase Roman
i	Lowercase Roman
omitted	Interpreted as 1 (Arabic)
illegal	No effect

By default, the complete heading mark for a given level is built by concatenating the mark for that level to the right of all marks for all levels of higher value. To inhibit the concatenation of heading level marks, i.e., to obtain just the current level mark followed by a period, the heading mark type register **(Ht)** is set to 1. For example, input for a commonly used "outline" style is:

```
.HM I A 1 a i
.nr Ht 1
```

4.3 Unnumbered Headings

.HU *heading-text*

The **.HU** macro is a special case of **.H**; it is handled in the same way as **.H** except that no heading mark is printed. In order to preserve the hierarchical structure of headings when **.H** and **.HU** calls are intermixed, each **.HU** heading is considered to exist at the level given by register **Hu**, whose initial value is 2. Thus, in the normal case, the only difference between:

.HU *heading-text*

and

.H 2 *heading-text*

is the printing of the heading mark for the latter. Both macros have the effect of incrementing the numbering counter for level 2 and

resetting to zero the counters for levels 3 through 7. Typically, the value of **Hu** should be set to make unnumbered headings (if any) be the lowest-level headings in a document.

The **.HU** macro can be especially helpful in setting up appendices and other sections that may not fit well into the numbering scheme of the main body of a document {14.2.1}.

4.4 Headings and Table of Contents

The text of headings and their corresponding page numbers can be automatically collected for a table of contents. This is accomplished by doing the following:

- specifying in the contents level register, **Cl**, what level headings are to be saved
- invoking the **.TC** macro {10.1} at the end of the document.

Any heading whose level is less than or equal to the value of the **Cl** register is saved and later displayed in the table of contents. The default value for the **Cl** register is 2, i.e., the first two levels of headings are saved.

Due to the way headings are saved, it is possible to exceed the formatter's storage capacity, particularly when saving many levels of many headings, while also processing displays {7} and footnotes {8}. If this happens, the "Out of temp file space" formatter error message (Table 16.D) will be issued; the only remedy is to save fewer levels and/or to have fewer words in the heading text.

4.5 First-Level Headings and Page Numbering Style

By default, pages are numbered sequentially at the top of the page. For large documents, it may be desirable to use page numbering of the "section-page" form where "section" is the number of the current first-level heading. This page numbering style can be achieved by specifying the **−rN3** or **−rN5** flag on the command line {9.3}. As a side effect, this also has the effect of setting **Ej** to 1, which causes each first-level section to begin on a new page. In this style, the page number is printed at the bottom of the page so that the correct section number is printed.

4.6 User Exit Macros

Note: This paragraph is intended primarily for users who are accustomed to writing formatter macros.

.HX *dlevel rlevel heading-text*
.HY *dlevel rlevel heading-text*
.HZ *dlevel rlevel heading-text*

The **.HX**, **.HY**, and **.HZ** macros are the means by which the user obtains a final level of control over the previously described heading mechanism. These macros are not defined by **mm**, they are intended to be defined by the user. The **.H** macro call invokes **.HX** shortly before the actual heading text is printed; it calls **.HZ** as its last action. After **.HX** is invoked, the size of the heading is calculated. This processing causes certain features that may have been included in **.HX**, such as **.ti** for temporary indent, to be lost. After the size calculation, **.HY** is invoked so that the user may respecify these features. All default actions occur if these macros are not defined. If **.HX**, **.HY**, or **.HZ** are defined by the user, user-supplied definition is interpreted at the appropriate point. These macros can therefore influence handling of all headings because the **.HU** macro is actually a special case of the **.H** macro.

If the user originally invoked the **.H** macro, then the derived level argument (*dlevel*) and the real level argument (*rlevel*) are both equal to the level given in the **.H** invocation. If the user originally invoked the **.HU** macro {4.3}, *dlevel* is equal to the contents of register **Hu**, and *rlevel* is 0. In both cases, *heading-text* is the text of the original invocation.

By the time **.H** calls **.HX**, it has already incremented the heading counter of the specified level, produced blank lines (vertical spaces) to precede the heading {4.2.2.1}, and accumulated the "heading mark", i.e., the string of digits, letters, and periods needed for a numbered heading. When **.HX** is called, all user-accessible registers and strings can be referenced, as well as the following:

string }0 — If *rlevel* is nonzero, this string contains the "heading mark". Two unpaddable spaces (to separate the *mark* from the *heading*) have been appended to this string. If *rlevel* is 0, this string is null.

register ;0 This register indicates the type of spacing that is to follow the heading {4.2.2.2}.
A value of 0 means that the heading is run-in.
A value of 1 means a break (but no blank line) is to follow the heading.
A value of 2 means that a blank line (**nroff**) or one-half a vertical space (**troff**) is to follow the heading.

string }2 If "register ;0" is 0, this string contains two unpaddable spaces that will be used to separate the (run-in) heading from the following text.
If "register ;0" is nonzero, this string is null.

register ;3 This register contains an adjustment factor for a **.ne** request issued before the heading is actually printed. On entry to **.HX**, it has the value 3 if *dlevel* equals 1, and a value of 1 otherwise. The **.ne** request is for the following number of lines: the contents of the "register ;0" taken as blank lines (**nroff**) or halves of vertical space (**troff**) plus the contents of "register ;3" as blank lines (**nroff**) or halves of vertical space (**troff**) plus the number of lines of the heading.

The user may alter the values of }0, }2, and ;3 within **.HX**. The following are examples of actions that might be performed by defining **.HX** to include the lines shown:

- Change first-level heading mark from format *n.* to *n*.0:

```
.if \\$1=1 .ds }0 \\n(H1.0\<sp>\<sp>
```

(where <sp> stands for a space)

- Separate run-in heading from the text with a period and two unpaddable spaces:

```
.if \\n(;0=0 .ds }2 .\<sp>\<sp>
```

- Assure that at least 15 lines are left on the page before printing a first-level heading:

```
.if \\$1=1 .nr ;3 (15-\\n(;0)v
```

- Add three additional blank lines before each first-level heading:

```
.if \\$1=1 .sp 3
```


- Indent level 3 run-in headings by five spaces:

```
.if \\$1=3 .ti 5n
```

If temporary strings or macros are used within **.HX**, their names should be chosen with care {14.1}.

When the **.HY** macro is called after the **.ne** is issued, certain features requested in **.HX** must be repeated. For example:

```
.de HY
.if \\$1=3 .ti 5n
..
```

The **.HZ** macro is called at the end of **.H** to permit user-controlled actions after the heading is produced. In a large document, sections may correspond to chapters of a book; and the user may want to change a page header or footer, e.g.:

```
.de HZ
.if \\$1=1 .PF "Section \\$3"
..
```

4.7 Hints for Large Documents

A large document is often organized for convenience into one input text file per section. If the files are numbered, it is wise to use enough digits in the names of these files for the maximum number of sections, i.e., use suffix numbers 01 through 20 rather than 1 through 9 and 10 through 20.

Users often want to format individual sections of long documents. To do this with the correct section numbers, it is necessary to set register **H1** to one less than the number of the section just before the corresponding **.H 1** call. For example, at the beginning of Part 5, insert

```
.nr H1 4
```

It will also be necessary to set the correct page number by using the **.pn** request or the **−rP***n* flag.

Note: This is not good practice. It defeats the automatic (re)numbering of sections when sections are added or deleted. Such lines should be removed as soon as possible.

5. Lists

This part describes different styles of lists; automatically numbered and alphabetized lists, bullet lists, dash lists, lists with arbitrary marks, and lists starting with arbitrary strings, i.e., with terms or phrases to be defined.

5.1 List Spacing

Spacing at the beginning of the list and between items can be suppressed by setting the list space register (**Ls**). The **Ls** register is set to the innermost list level for which spacing is done. For example:

```
.nr Ls 0
```

specifies that no spacing will occur around any list items. The default value for **Ls** is six (which is the maximum list nesting level).

5.2 List Macros

In order to avoid repetitive typing of arguments to describe the style or appearance of items in a list, **mm** provides a convenient way to specify lists. All lists share the same overall structure and are composed of the following basic parts:

- A *list-initialization macro* (**.AL .BL, .DL, .ML, .RL,** or **.VL**) determines the style of the list: line spacing, indentation, marking with special symbols, and numbering or alphabetizing of list items.
- One or more *list-item macros* (**.LI**) identifies each unique item to the system. It is followed by the actual text of the corresponding list item.
- The *list-end macro* (**.LE**) identifies the end of the list. It terminates the list and restores the previous indentation.

Lists may be nested up to six levels. The list-initialization macro saves the previous list status (indentation, marking, style, etc.); the **.LE** macro restores it.

With this approach, the format of a list is specified only once at the beginning of the list. In addition, by building onto the existing structure, users may create their own customized sets of list macros with relatively little effort ({5.12} and {5.13}).

5.2.1 List-Initialization Macros

List-initialization macros are implemented as calls to the more basic **.LB** macro {5.12}. They are:

.AL Automatically Numbered or Alphabetized List
.BL Bullet List
.DL Dash List
.ML Marked List
.RL Reference List
.VL Variable-Item List

5.3 Automatically Numbered or Alphabetized List

.AL [*type*] [*text-indent*] [1]

The **.AL** macro is used to begin sequentially numbered or alphabetized lists. If there are no arguments, the list is numbered; and text is indented by **Li** (default is six) spaces from the indent in force when the **.AL** is called. This leaves room for a space, two digits, a period, and two spaces before the text. Values that specify indentation must be unscaled and are treated as "character positions", i.e., number of ens. (The string **.AL A 5** is used to initialize the following list.)

A. The *type* argument may be given to obtain a different type of sequencing. Its value indicates the first element in the sequence desired. If *type* argument is omitted or null, the value 1 is assumed.

ARGUMENT	INTERPRETATION
1	Arabic (default for all levels)
A	Uppercase alphabetic
a	Lowercase alphabetic
I	Uppercase Roman
i	Lowercase Roman

B. If *text-indent* argument is non-null, it is used as the number of spaces from the current indent to the text, i.e., it is used instead of the **Li** register for this list only. If *text-indent* argument is null, the value of **Li** will be used.

C. If the third argument is given, a blank line (**nroff**) or one-half a vertical space (**troff**) will not separate items in the list. A blank line will occur before the first item however.

5.4 Bullet List

.BL [*text-indent*] [1]

The **.BL** macro begins a bullet list. Each list item is marked by a bullet (•) followed by one space. (The string **.BL 5** is used to initialize the following list.)

- If the *text-indent* argument is specified (non-null), it overrides the default indentation which is the amount of paragraph indentation as given in the **Pi** register {4.1}. In the default case, the text of a bullet list lines up with the first line of indented paragraphs.
- If the second argument is specified, no blank lines will separate items in the list.

5.5 Dash List

.DL [*text-indent*] [1]

The **.DL** macro begins a dash list. Each list item is marked by a dash (—) followed by one space. (The string **.DL 5** is used to initialize the following list.)

— If the *text-indent* argument is specified (non-null), it overrides the default indentation which is the amount of paragraph indentation as given in the **Pi** register {4.1}. In the default case, the text of a dash list lines up with the first line of indented paragraphs.

— If the second argument is specified, no blank lines will separate items in the list.

5.6 Marked List

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

The **.ML** macro is much like **.BL** and **.DL** macros but expects the user to specify an arbitrary *mark* which may consist of more than a single character. (The string **.ML \(sq 5** is used to initialize the following list.)

□ Text is indented *text-indent* spaces if the second argument is specified (non-null); otherwise, the text is indented one more space than the width of *mark*.

□ If the third argument is specified, no blank lines will separate items in the list.

Note: The *mark* must not contain ordinary (paddable) spaces because alignment of items will be lost if the right margin is justified {3.3}.

5.7 Reference List

.RL [*text-indent*] [1]

A **.RL** macro call begins an automatically numbered list in which the numbers are enclosed by square brackets ([ ]). (The string **.RL 5** is used to initialize the following list.)

[1] If *text-indent* argument is specified (non-null), it is used as the number of spaces from the current indent to the text, i.e., it is used instead of **Li** for this list only. If *text-indent* argument is omitted or null, the value of **Li** is used.

[2] If the second argument is specified, no blank lines will separate the items in the list.

5.8 Variable-Item List

.VL *text-indent* [*mark-indent*] [1]

When a list begins with a **.VL** macro, there is effectively no current *mark*; it is expected that each **.LI** will provide its own mark. This form is typically used to display definitions of terms or phrases.

- *Text-indent* provides the distance from current indent to beginning of the text.
- *Mark indent* produces the number of spaces from current indent to beginning of the *mark*, and it defaults to 0 if omitted or null.
- If the third argument is specified, no blank lines will separate items in the list.

An example of **.VL** macro usage is shown below:

```
.VL 20 5
.LI "First\ Mark"
This is the first mark specified for this list.
.LI "Second\ Mark"
.br
This is the second mark specified for this list.
The \fB.br\fR request causes a break so that this
text will appear one line below the mark.
.LI "Third\ Mark\ Longer\ Than\ Indent:"
This item shows the effect of a long mark;
one space separates the mark from the text.
.LI "\ "
This item has a nonprinting mark and effectively
produces a list item that is indented.
.LI
This item has an omitted mark and produces a "hanging indent".
The first line of text is at the left margin and
the second is indented.
.LE
```

when formatted yields:

First Mark This is the first mark specified for this list.

Second Mark
This is the second mark specified for this list. The **.br** request causes a break so that this text appears one line below the mark.

Third Mark Longer Than Indent: This item shows the effect of a long mark; one space separates the mark from the text.

This item has a nonprinting mark (an unpaddable space) and effectively produces a list item that is indented.

This item has an omitted mark and produces a "hanging indent". The first line of text is at the left margin and the second is indented.

Note: The *mark* must not contain ordinary (paddable) spaces because alignment of items will be lost if the right margin is justified {3.3}.

5.9 List-Item Macro

.LI [*mark*] [1]
one or more lines of text that make up the list item.

The **.LI** macro is used with all lists and for each list item. It normally causes output of a single blank line (**nroff**) or one-half a vertical space (**troff**) before its list item although this may be suppressed.

- If no arguments are given, **.LI** labels the item with the current *mark* (except in **.VL** lists) which is specified by the most recent list-initialization macro.
- If a single argument is given, that argument is output instead of the current *mark*.
- If two arguments are given, the first argument becomes a prefix to the current *mark* thus allowing the user to emphasize one or more items in a list. One unpaddable space is inserted between the prefix and the mark.

For example:

```
.BL 5
.LI
This is a simple bullet item.
.LI +
This replaces the bullet with a "plus".
.LI + 1
This uses a "plus" as prefix to the bullet.
.LE
```

when formatted yields:

- This is a simple bullet item.

\+ This replaces the bullet with a "plus".

\+ • This uses a "plus" as prefix to the bullet.

Note: The *mark* must not contain ordinary (paddable) spaces because alignment of items will be lost if the right margin is justified {3.3}.

If the current *mark* (in the current list) is a null string and the first argument of **.LI** is omitted or null, the resulting effect is that of a "hanging indent", i.e., the first line of the following text is moved to

the left starting at the same place where *mark* would have started {5.8}.

5.10 List-End Macro

.LE [1]

The **.LE** macro restores the state of the list to that existing just before the most recent list-initialization macro call. If the optional argument is given, the **.LE** outputs a blank line (**nroff**) or one-half a vertical space (**troff**). This option should generally be used only when the **.LE** is followed by running text but not when followed by a macro that produces blank lines of its own such as the **.P** or **.H** macro.

The **.H** and **.HU** macros automatically clear all list information. The user may omit the **.LE** macros that would normally occur just before either of these macros and not receive the "LE:mismatched" error message. Such a practice is not recommended because errors will occur if the list text is separated from the heading at some later time (e.g., by insertion of text).

5.11 Example of Nested Lists

An example of input for the several lists and the corresponding output is shown below. The **.AL** and **.DL** macro calls {5.3, 5.5} contained therein are examples of list-initialization macros. Input text is:


```
.AL A 5
.LI
This is automatically alphabetized list item A.
This list item has an indentation of 5 ens.
.AL
.LI
This is automatically numbered list item 1.
This list item also has an indentation of 5 ens.
.DL
.LI
This is a dash list item.
.LI + 1
This is another dash item in the same list
as the above item with a "plus" as prefix.
This is the last item in the dash list.
.LE
.LI
This is item 2 in the automatically numbered list.
This is the last item in the automatically numbered list.
.LE
.LI
This is item B in the automatically alphabetized list.
This is the last item in the automatically numbered list.
.LE
```

The output is:

A. This is automatically alphabetized list item A. This list item has an indentation of 5 ens.

   1. This is automatically numbered list item 1. This list item also has an indentation of 5 ens.

      – This is a dash list item.

      + – This is another dash item in the same list as the above item with a "plus" as prefix. This is the last item in the dash list.

   2. This is item 2 in the automatically numbered list. This is the last item in the automatically numbered list.

B. This is item B in the automatically alphabetized list. This is the last item in the automatically numbered list.

5.12 List-Begin Macro and Customized Lists

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

List-initialization macros described above suffice for almost all cases. However, if necessary, the user may obtain more control over the layout of lists by using the basic list-begin macro (**.LB**). The **.LB** macro is used by the other list-initialization macros. Its arguments are as follows:

- The *text-indent* argument provides the number of spaces that text is to be indented from the current indent. Normally, this value is taken from the **Li** register (for automatic lists) or from the **Pi** register (for bullet and dash lists).
- The combination of *mark-indent* and *pad* arguments determines the placement of the mark. The mark is placed within an area (called *mark area*) that starts *mark-indent* spaces to the right of the current indent and ends where the text begins (i.e., ends *text-indent* spaces to the right of the current indent). The *mark-indent* argument is typically 0.
- Within the *mark area*, the mark is left justified if the *pad* argument is 0. If *pad* is a number *n* (greater than 0) then *n* blanks are appended to the mark; the *mark-indent* value is ignored. The resulting string immediately precedes the text. The *mark* is effectively right justified *pad* spaces immediately to the left of text.
- Arguments *type* and *mark* interact to control the type of marking used. If *type* is 0, simple marking is performed using the mark character(s) found in the *mark* argument. If *type* is greater than 0, automatic numbering or alphabetizing is done; and *mark* is then interpreted as the first item in the sequence to be used for numbering or alphabetizing, i.e., it is chosen from the set (1, A, a, I, i) as in {5.3}. This is summarized in the following table:

ARGUMENT		RESULT
type	**mark**	
0	omitted	hanging indent
0	*string*	*string* is the mark
>0	omitted	Arabic numbering
>0	one of: 1, A, a, I, i	automatic numbering or alphabetic sequencing

Each nonzero value of *type* from one to six selects a different way of displaying the marks. The following table shows the output appearance for each value of *type*:

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

where *x* is the generated number or letter.

Note: The *mark* must not contain ordinary (paddable) spaces because alignment of items will be lost if the right margin is justified {3.3}.

- The *LI-space* argument gives the number of blank lines (**nroff**) or halves of a vertical space (**troff**) that should be output by each **.LI** macro in the list. If omitted, *LI-space* defaults to 1; the value 0 can be used to obtain compact lists. If *LI-space* is greater than 0, the **.LI** macro issues a **.ne** request for two lines just before printing the mark.
- The *LB-space* argument is the number of blank lines (**nroff**) or half vertical spaces (**troff**) to be output by **.LB** itself. If omitted *LB-space* defaults to 0.

There are three combinations of *LI-space* and *LB-space*:

- The normal case is to set *LI-space* to 1 and *LB-space* to 0 yielding one blank line (**nroff**) or one-half a vertical space (**troff**) before each item in the list; such a list is usually terminated with a **.LE 1** macro to end the list with a blank line (**nroff**) or one-half a vertical space (**troff**).
- For a more compact list, *LI-space* is set to 0, *LB-space* is set to 1, and the **.LE 1** macro is used at the end of the list. The result is a list with one blank line (**nroff**) or one-half a vertical space (**troff**) before and after it.
- If both *LI-space* and *LB-space* are set to 0 and the **.LE** macro is used to end the list, a list without any blank lines will result.

Section 5.13 shows how to build upon the supplied list of macros to obtain other kinds of lists.

5.13 User-Defined List Structures

Note: This part is intended for users accustomed to writing formatter macros.

If a large document requires complex list structures, it is useful to define the appearance for each list level only once instead of having to define the appearance at the beginning of each list. This permits consistency of style in a large document. A generalized list-initialization macro might be defined in such a way that what the macro does depends on the list-nesting level in effect at the time the macro is called. Levels 1 through 5 of the lists to be formatted may have the following appearance:

A.

[1]

•

a)

+

The following code defines a macro (**.aL**) that always begins a new list and determines the type of list according to the current list level. To understand it, the user should know that the number register **:g** is used by the **mm** list macros to determine the current list level; it is 0 if there is no currently active list. Each call to a list-initialization macro increments **:g**, and each **.LE** call decrements it.

```
\" register g is used as a local temporary to save
\" :g before it is changed below
.de aL
.nr g \\n(:g
.if \\ng=0 .AL A                    \" produces an A.
.if \\ng=1 .LB \\n(Li 0 1 4         \" produces a [1]
.if \\ng=2 .BL                      \" produces a bullet
.if \\ng=3 .LB \\n(Li 0 2 2 a       \" produces an a)
.if \\ng=4 .ML +                    \" produces a +
..
```


This macro can be used (in conjunction with **.LI** and **.LE**) instead of **.AL**, **.RL**, **.BL**, **.LB**, and **.ML**. For example, the following input:

```
.al
.LI
First line.
.aL
.LI
Second line.
.LE
.LI
Third line.
.LE
```

when formatted yields

1. First line.

   [1] Second line.

2. Third line.

There is another approach to lists that is similar to the **.H** mechanism. List-initialization, as well as the **.LI** and the **.LE** macros, are all included in a single macro. That macro (defined as **.bL** below) requires an argument to tell it what level of item is required; it adjusts the list level by either beginning a new list or setting the list level back to a previous value, and then issues a **.LI** macro call to produce the item:

```
.de bL
.ie \\n(.$ .nr g \\$1               \" if there is an argument, that
                                    is the level
.el .nr g \\n(:g                    \" if no argument, use current
                                    level
.if \\ng-\\n(:g>1 .)D               \" **ILLEGAL SKIPPING OF
                                    LEVEL
.\"                                     increasing level by more
                                    than 1
.if \\ng>\\n(:g \{.aL \\ng-1         \" if g > :g, begin new list
.nr                                 \" and reset g to current level
.\"                                     (.aL changes g)
.if \\n(:g>\\ng .LC \\ng            \" if :g > g, prune back to
                                    correct level
```

```
.\" if :g = g, stay within current list
.LI                                    \" in all cases, get out an item
..
```

For **.bL** to work, the previous definition of the **.aL** macro must be changed to obtain the value of **g** from its argument rather than from **:g**. Invoking **.bL** without arguments causes it to stay at the current list level. The **.LC** (list clear) macro removes list descriptions until the level is less than or equal to that of its argument. For example, the **.H** macro includes the call "**.LC 0**". If text is to be resumed at the end of a list, insert the call "**.LC 0**" to clear out the lists completely. The example below illustrates the relatively small amount of input needed by this approach. The input text

```
The quick brown fox jumped over the lazy dog's back.
.bL 1
First line.
.bL 2
Second line.
.bL 1
Third line.
.bL
Fourth line.
.LC 0
Fifth line.
```

when formatted yields

The quick brown fox jumped over the lazy dog's back.

A. First line.

   [1] Second line.

B. Third line.

C. Fourth line.

Fifth line.

6. Memorandum and Released-Paper Style Documents

Note: Some of the information in this section is applicable for Bell Laboratories only. However, most of the features discussed here can be tailored to specific needs.

One use of the Memorandum Macros is for the preparation of memoranda and released-paper documents which have special requirements for the first page and for the cover sheet. Data needed (title, author, date, case numbers, etc.) is entered the same for both styles; an argument to the **.MT** macro indicates which style is being used.

6.1 Sequence of Beginning Macros

Macros, if present, must be given in the following order:

```
.ND new-date
.TL [charging-case] [filing-case]
one or more lines of text
.AF [company-name]
.AU name [initials] [loc] [dept] [ext] [room] [arg] [arg]
.AT [title] ...
.TM [number] ...
.AS [arg] [indent]
one or more lines of abstract text
.AE
.NS [arg]
one or more lines of "Copy to" notation
.NE
.OK [keyword] ...
.MT [type] [addressee]
```

The only required macros for a memorandum for file or a released-paper document are **.TL**, **.AU**, and **.MT**; all other macros (and their associated input lines) may be omitted if the features are not needed. Once **.MT** has been invoked, none of the above macros (except **.NS** and **.NE**) can be reinvoked because they are removed from the table of defined macros to save memory space.

If neither the memorandum nor released-paper style is desired, the **.TL**, **.AU**, **.TM**, **.AE**, **.OK**, **.MT**, **.ND**, and **.AF** macros should be omitted from the input text. If these macros are omitted, the first page will have only the page header followed by the body of the document.

6.2 Title

```
.TL [charging-case] [filing-case]
one or more lines of title text
```

Arguments to the **.TL** macro are the charging-case number(s) and filing-case number(s).

- The *charging-case* argument is the case number to which time was charged for the development of the project described in the memorandum. Multiple charging-case numbers are entered as "subarguments" by separating each from the previous with a comma and a space and enclosing the entire argument within double quotes.
- The *filing-case* argument is a number under which the memorandum is to be filed. Multiple filing case numbers are entered similarly. For example:

```
.TL "12345, 67890" 987654321
Construction of a Table of All Even Prime Numbers
```

The title of the memorandum or released-paper document follows the **.TL** macro and is processed in fill mode. The **.br** request may be used to break the title into several lines as follows:

```
.TL 12345
First Title Line
.br
\!.br
Second Title Line
```

On output, the title appears after the word "Subject" in the memorandum style and is centered and bold in the released-paper document style.

If only a charging case number or only a filing case number is given, it will be separated from the title in the memorandum style by a dash and will appear on the same line as the title. If both case numbers are given and are the same, then "Charging and Filing Case" followed by the number will appear on a line following the title. If the two case numbers are different, separate lines for "Charging Case" and "File Case" will appear after the title.

6.3 Authors

.AU *name* [*initials*] [*loc*] [*dept*] [*ext*] [*room*] [*arg*] [*arg*]
.AT [*title*] ...

The **.AU** macro receives as arguments information that describes an author. If any argument contains blanks, that argument must be enclosed within double quotes. The first six arguments must appear in the order given. A separate **.AU** macro is required for each author.

The **.AT** macro is used to specify the author's title. Up to nine arguments may be given. Each will appear in the signature block for memorandum style {6.11} on a separate line following the signer's name. The **.AT** must immediately follow the **.AU** for the given author. For example:

```
.AU "S. J. Jones" JJJ PY 9876 5432 1Z-234
.AT Director "Materials Research Laboratory"
```

In the "From" portion in the memorandum style, the author's name is followed by location and department number on one line and by room number and extension number on the next line. The "x" for the extension is added automatically. Printing of the location, department number, extension number, and room number may be suppressed on the first page of a memorandum by setting the register **Au** to 0; the default value for **Au** is 1. Arguments 7 through 9 of the **.AU** macro, if present, will follow this normal author information in the "From" portion, each on a separate line. These last three arguments may be used for organizational numbering schemes, etc. For example:

```
.AU "S. P. Lename" SPL IH 9988 7766 5H-444 9876-543210.01MF
```

The name, initials, location, and department are also used in the signature block. Author information in the "From" portion, as well as names and initials in the signature block will appear in the same order as the **.AU** macros.

Note: Names of authors in the released-paper style are centered below the title. Following the name of the last author, "Bell Laboratories" and the location are centered. The paragraph on memorandum types {6.7} contains information regarding authors from different locations.

6.4 TM Numbers

.TM [*number*] ...

If the memorandum is a technical memorandum, the TM numbers are supplied via the **.TM** macro. Up to nine numbers may be specified. For example:

```
.TM 7654321 77777777
```

This macro call is ignored in the released-paper and external-letter styles {6.7}.

6.5 Abstract

.AS [*arg*] [*indent*]
text of abstract
.AE

If a memorandum has an abstract, the input is identified with the **.AS** (abstract start) and **.AE** (abstract end) delimiters. Abstracts are printed on page 1 of a document and/or on its cover sheet. There are three styles of cover sheet:

- released paper
- technical memorandum
- memorandum for file (also used for engineer's notes, memoranda for record, etc.)

Cover sheets for released papers and technical memoranda are obtained by invoking the **.CS** macro {10.2}.

In released-paper style (first argument of the **.MT** macro {6.7} is 4) and in technical memorandum style if the first argument of **.AS** is:

0 — Abstract will be printed on page 1 and on the cover sheet (if any).

1 — Abstract will appear only on the cover sheet (if any).

In memoranda for file style and in all other documents (other than external letters), if the first argument of **.AS** is:

0 — Abstract will appear on page 1 and there will be no cover sheet printed.

2 — Abstract will appear only on the cover sheet which will be produced automatically (i.e., without invoking the **.CS** macro).

It is not possible to get either an abstract or a cover sheet with an external letter (first argument of the **.MT** macro is 5).

Notations such as a "Copy to" list {6.11.2} are allowed on memorandum for file cover sheets; the **.NS** and **.NE** macros must appear after the **.AS 2** and **.AE** macros. Headings {4.2, 4.3} and displays {7} are not permitted within an abstract.

The abstract is printed with ordinary text margins; an indentation to be used for both margins can be specified as the second argument of **.AS**. Values that specify indentation must be unscaled and are treated as "character positions", i.e., as the number of *ens*.

6.6 Other Keywords

.OK [*keyword*] ...

Topical keywords should be specified on a technical memorandum cover sheet. Up to nine such keywords or keyword phrases may be specified as arguments to the **.OK** macro; if any keyword contains spaces, it must be enclosed within double quotes.

6.7 Memorandum Types

.MT [*type*] [*addressee*]

The **.MT** macro controls the format of the top part of the first page of a memorandum or of a released-paper document and the format of the cover sheets. The *type* arguments and corresponding values are:

type	VALUE
""	no memorandum type printed
0	no memorandum type printed
none	MEMORANDUM FOR FILE
1	MEMORANDUM FOR FILE
2	PROGRAMMER'S NOTES
3	ENGINEER'S NOTES
4	released-paper style

5	external-letter style
"*string*"	*string* (enclosed in quotes)

If the *type* argument indicates a memorandum style document, the corresponding statement indicated under "VALUE" will be printed after the last line of author information. If *type* is longer than one character, then the string, itself, will be printed. For example:

```
.MT "Technical Note #5"
```

A simple letter is produced by calling **.MT** with a null (but not omitted) or 0 argument.

The second argument to **.MT** is the name of the addressee of a letter. If present, that name and the page number replace the normal page header on the second and following pages of a letter. For example:

```
.MT 1 "Steve Jones"
```

produces

```
Steve Jones – 2
```

The *addressee* argument may not be used if the first argument is 4 (released-paper style document).

The released-paper style is obtained by specifying

```
.MT 4 [1]
```

This results in a centered, bold title followed by centered names of authors. The location of the last author is used as the location following "Bell Laboratories" (unless the **.AF** macro specifies a different company). If the optional second argument to **.MT** 4 is given, then the name of each author is followed by the respective company name and location. Information necessary for the memorandum style document but not for the released-paper style document is ignored.

If the released-paper style document is utilized, most BTL location codes are defined as strings that are the addresses of the corresponding BTL locations. These codes are needed only until the **.MT** macro is invoked. Thus, following the **.MT** macro, the user may reuse these string names. In addition, the macros for the end of a memorandum {6.11} and their associated lines of input are ignored when the

released-paper style is specified.

Authors from non-BTL locations may include their affiliations in the released-paper style by specifying the appropriate **.AF** macro {6.9} and defining a string (with a 2-character name such as ZZ) for the address before each **.AU**. For example:

```
.TL
A Learned Treatise
.AF "Getem Inc."
.ds ZZ "22 Maple Avenue, Sometown 09999"
.AU "F. Swatter" "" ZZ
.AF "Bell Laboratories"
.AU "Sam P. Lename" "" CB
.MT 4 1
```

In the external-letter style document (**.MT 5**), only the title (without the word "Subject:") and the date are printed in the upper left and right corners, respectively, on the first page. It is expected that preprinted stationery will be used with the company logo and address of the author.

6.8 Date Changes

.ND *new-date*

The **.ND** macro alters the value of the string **DT**, which is initially set to produce the current date. If the argument contains spaces, it must be enclosed within double quotes.

6.9 Alternate First-Page Format

.AF [*company-name*]

An alternate first-page format can be specified with the **.AF** macro. The words "Subject", "Date", and "From" (in the memorandum style) are omitted and an alternate company name is used.

If an argument is given, it replaces "Bell Laboratories" without affecting other headings. If the argument is null, "Bell Laboratories" is suppressed; and extra blank lines are inserted to allow room for

stamping the document with a Bell System logo or a Bell Laboratories stamp.

The **.AF** with no argument suppresses "Bell Laboratories" and the "Subject/Date/From" headings, thus allowing output on preprinted stationery. The use of **.AF** with no arguments is equivalent to the use of **−rA1** {2.4}, except that the latter must be used if it is necessary to change the line length and/or page offset (which default to 5.8i and 1i, respectively, for preprinted forms). The command line options **−rOk** and **−rWk** {2.4} are not effective with **.AF**. The only **.AF** use appropriate for the **troff** formatter is to specify a replacement for "Bell Laboratories".

The command line option **−rE***n* {2.4} controls the font of the "Subject/Date/From" block.

6.10 Example

Input text for a document may begin as follows:

```
.TL
MM\*(EMMemorandum Macros
.AU "D. W. Smith" DWS PY
.AU "J. R. Mashey" JRM PY
.AU "E. C. Pariser (January 1980 Revision)" ECP PY
.AU "N. W. Smith (June 1980 Revision)" NWS PY
.MT 4
```

Figure 16.A at the end of this chapter shows the input text file and both the **nroff** and **troff** formatter outputs for a simple letter.

6.11 End of Memorandum Macros

At the end of a memorandum document (but not of a released-paper document), signatures of authors and a list of notations can be requested. The following macros and their input are ignored if the released-paper style document is selected.

6.11.1 Signature Block

.FC [*closing*]
.SG [*arg*] [1]

The **.FC** macro prints "Yours very truly," as a formal closing, if no closing argument is used. It must be given before the **.SG** macro. A different closing may be specified as an argument to **.FC**.

The **.SG** macro prints the author's name(s) after the formal closing, if any. Each name begins at the center of the page. Three blank lines are left above each name for the actual signature.

- If no arguments are given, the line of reference data (location code, department number, author's initials, and typist's initials, all separated by hyphens) will not appear.
- A non-null first argument is treated as the typist's initials and is appended to the reference data.
- A null first argument prints reference data without the typist's initials or the preceding hyphen.
- If there are several authors and if the second argument is given, reference data is placed on the line of the first author.

Reference data contains only the location and department number of the first author. Thus, if there are authors from different departments and/or from different locations, the reference data should be supplied manually after the invocation (without arguments) of the **.SG** macro. For example:

```
.SG
.rs
.sp −1v
PY/MH-9876/5432-JJJ/SPL-cen
```

6.11.2 "Copy to" and Other Notations

.NS [*arg*]
zero or more lines of the notation
.NE

Many types of notations (such as a list of attachments or "Copy to" lists) may follow signature and reference data. Various notations are obtained through the **.NS** macro, which provides for proper spacing and for breaking notations across pages, if necessary.

Codes for *arg* and the corresponding notations are:

arg	NOTATIONS
none	Copy to
""	Copy to
0	Copy to
1	Copy (with att.) to
2	Copy (without att.) to
3	Att.
4	Atts.
5	Enc.
6	Encs.
7	Under Separate Cover
8	Letter to
9	Memorandum to
"*string*"	Copy (string) to

If *arg* consists of more than one character, it is placed within parentheses between the words "Copy" and "to". For example:

.NS "with att. 1 only"

will generate

Copy (with att. 1 only) to

as the notation. More than one notation may be specified before the **.NE** macro because a **.NS** macro terminates the preceding notation, if any. For example:

.NS 4
Attachment 1-List of register names
Attachment 2-List of string and macro names
.NS 1
S. J. Jones
.NS 2
S. P. Lename
G. H. Hurtz
.NE

would be formatted as

Atts.
Attachment 1-List of register names
Attachment 2-List of string and macro names

Copy (with att.) to
S. J. Jones

Copy (without att.) to
S. P. Lename
G. H. Hurtz

The **.NS** and **.NE** macros may also be used at the beginning following **.AS 2** and **.AE** to place the notation list on the memorandum for file cover sheet {6.5}. If notations are given at the beginning without **.AS 2**, they will be saved and output at the end of the document.

6.11.3 Approval Signature Line

.AV *approver's-name*

The **.AV** macro may be used after the last notation block to automatically generate a line with spaces for the approval signature and date. For example:

.AV "Jane Doe"

produces

APPROVED:

_____ _____

Jane Doe Date

6.12 One-Page Letter

At times, the user may like more space on the page forcing the signature or items within notations to the bottom of the page so that the letter or memo is only one page in length. This can be accomplished by increasing the page length with the **−rL***n* option, e.g., **−rL90**. This has the effect of making the formatter believe that the page is 90 lines long and therefore providing more space than usual to place the signature or the notations.

7. Displays

Displays are blocks of text that are to be kept together on a page and not split across pages. They are processed in an environment that is different from the body of the text (see the **.ev** request in Chapter 3). The Memorandum Macros package provides two styles of displays – a *static* (**.DS**) style and a *floating* (**.DF**) style.

- In the *static* style, the display appears in the same relative position in the output text as it does in the input text. This may result in extra white space at the bottom of the page if the display is too long to fit in the remaining page space.
- In the *floating* style, the display "floats" through the input text to the top of the next page if there is not enough space on the current page. Thus input text that follows a floating display may precede it in the output text. A queue of floating displays is maintained so that their relative order of appearance in the text is not disturbed.

By default, a display is processed in no-fill mode with single spacing and is not indented from the existing margins. The user can specify indentation or centering as well as fill-mode processing.

Note: Displays and footnotes {8} may never be nested in any combination. Although lists {5} and paragraphs {4.1} are permitted, no headings (**.H** or **.HU**) {4.2, 4.3} can occur within displays or footnotes.

7.1 Static Displays

.DS [*format*] [*fill*] [*rindent*]
one or more lines of text
.DE

A static display is started by the **.DS** macro and terminated by the **.DE** macro. With no arguments, **.DS** accepts lines of text exactly as typed (no-fill mode) and will not indent lines from the prevailing left margin indentation or from the right margin.

- The *format* argument is an integer or letter used to control the left margin indentation and centering with the following meanings:

format	MEANING
""	no indent
omitted	no indent
0 or L	no indent
1 or I	indent by standard amount
2 or C	center each line
3 or CB	center as a block

- The *fill* argument is an integer or letter and can have the following meanings:

fill	MEANING
""	no-fill mode
omitted	no-fill mode
0 or N	no-fill mode
1 or F	fill mode

- The *rindent* argument is the number of characters that the line length should be decreased, i.e., an indentation from the right margin. This number must be unscaled in the **nroff** formatter and is treated as *ens*. It may be scaled in the **troff** formatter or else defaults to *ems*.

The standard amount of static display indentation is taken from the **Si** register, a default value of five spaces. Thus, text of an indented display aligns with the first line of indented paragraphs, whose indent is contained in the **Pi** register {4.1}. Even though their initial values are the same (default values), these two registers are independent.

The display *format* argument value 3 (or **CB**) centers (horizontally) the entire display as a block (as opposed to **.DS 2** and **.DF 2** which center each line individually). All collected lines are left justified, and the display is centered based on width of the longest line. This format must be used in order for the **eqn/neqn** "mark" and "lineup" feature to work with centered equations {7.4}.

By default, a blank line (**nroff**) or one-half a vertical space (**troff**) is placed before and after *static* and *floating* displays. These blank lines before and after *static* displays can be inhibited by setting the register **Ds** to 0.

The following example shows usage of all three arguments for *static* displays. This block of text will be indented five spaces (ems in **troff**) from the left margin, filled, and indented five spaces (ems in **troff**) from the right margin (i.e., centered). The input text

```
.DS I F 5
"We the people of the United States,
in order to form a more perfect union,
establish justice, ensure domestic tranquillity,
provide for the common defense,
and secure the blessings of liberty to
ourselves and our posterity,
do ordain and establish this Constitution to the
United States of America."
.DE
```

produces the output:

> "We the people of the United States, in order to form a more perfect union, establish justice, ensure domestic tranquillity, provide for the common defense, and secure the blessings of liberty to ourselves and our posterity, do ordain and establish this Constitution to the United States of America."

7.2 Floating Displays

.DF [*format*] [*fill*] [*rindent*]
one or more lines of text
.DE

A floating display is started by the **.DF** macro and terminated by the **.DE** macro. Arguments have the same meanings as static displays described above, except indent, no indent, and centering are calculated with respect to the initial left margin. This is because prevailing indent may change between when the formatter first reads the floating display and when the display is printed. One blank line (**nroff**) or one-half a vertical space (**troff**) occurs before and after a floating display.

The user may exercise precise control over the output positioning of floating displays through the use of two number registers, **De** and **Df** (see below). When a floating display is encountered by the **nroff** or **troff** formatter, it is processed and placed onto a queue of displays

waiting to be output. Displays are removed from the queue and printed in the order entered, which is the order they appeared in the input file. If a new floating display is encountered and the queue of displays is empty, the new display is a candidate for immediate output on the current page. Immediate output is governed by size of display and the setting of the **Df** register code. The **De** register code controls whether text will appear on the current page after a floating display has been produced.

As long as the display queue contains one or more displays, new displays will be automatically entered there, rather than being output. When a new page is started (or the top of the second column when in 2-column mode), the next display from the queue becomes a candidate for output if the **Df** register code has specified "top-of-page" output. When a display is output, it is also removed from the queue.

When the end of a section (using section-page numbering) or the end of a document is reached, all displays are automatically removed from the queue and output. This occurs before a **.SG**, **.CS**, or **.TC** macro is processed.

A display will fit on the current page if there is enough room to contain the entire display or if the display is longer than one page in length and less than half of the current page has been used. A wide (full-page width) display will not fit in the second column of a 2-column document.

The **De** and **Df** number register code settings and actions are as follows:

De REGISTER

CODE	ACTION
0	No special action occurs (also the default condition).
1	A page eject will always follow the output of each floating display, so only one floating display will appear on a page and no text will follow it.

Note: For any other code, the action performed is the same as for code 1.

Df REGISTER

CODE	ACTION
0	Floating displays will not be output until end of section (when section-page numbering) or end of document.
1	Output new floating display on current page if there is space; otherwise, hold it until end of section or document.
2	Output exactly one floating display from queue to the top of a new page or column (when in 2-column mode).
3	Output one floating display on current page if there is space; otherwise, output to the top of a new page or column.
4	Output as many displays as will fit (at least one) starting at the top of a new page or column.
0	If the *De* register is set to 1, each display will be followed by a page eject causing a new top of page to be reached where at least one more display will be output (this also applies to code 5).
5	Output a new floating display on the current page if there is room (default condition). Output as many displays (but at least one) as will fit on the page starting at the top of a new page or column.
	Note: For any code greater than 5, the action performed is the same as for code 5. If the **De** register is set to 1, each display will be followed by a page eject causing a new top of page to be reached where at least one more display will be output.

The **.WC** macro {12.4} may also be used to control handling of displays in double-column mode and to control the break in text before floating displays.

7.3 Tables

.TS [*H*]
global options;
column descriptors.
title lines
[**.TH** [*N*]]
data within the table.
.TE

The **.TS** (table start) and **.TE** (table end) macros make possible the use of the **tbl**(1) program. These macros are used to delimit text to be examined by **tbl** and to set proper spacing around the table. The display function and the **tbl** delimiting function are independent. In order to permit the user to keep together blocks that contain any mixture of tables, equations, filled text, unfilled text, and caption lines, the **.TS/.TE** block should be enclosed within a display (**.DS/.DE**). Floating tables may be enclosed inside floating displays (**.DF/.DE**).

Macros **.TS** and **.TE** permit processing of tables that extend over several pages. If a table heading is needed for each page of a multipage table, the "**H**" argument should be specified to the **.TS** macro as above. Following the options and format information, table title is typed on as many lines as required and is followed by the **.TH** macro. The **.TH** macro must occur when "**.TS H**" is used for a multipage table. This is not a feature of **tbl** but of the definitions provided by the Memorandum Macros package.

The **.TH** (table header) macro may take as an argument the letter **N**. This argument causes the table header to be printed only if it is the first table header on the page. This option is used when it is necessary to build long tables from smaller **.TS H/.TE** segments. For example:

```
.TS H
global options;
column descriptors.
Title lines
.TH
data
.TE
.TS H
global options;
column descriptors.
Title lines
.TH N
data
.TE
```

will cause the table heading to appear at the top of the first table segment and no heading to appear at the top of the second segment when both appear on the same page. However, the heading will still appear at the top of each page that the table continues onto. This feature is used when a single table must be broken into segments because of table complexity (e.g., too many blocks of filled text). If each segment had its own **.TS H/.TH** sequence, it would have its own header. However, if each table segment after the first uses **.TS H/.TH N**, the table header will appear only at the beginning of the table and the top of each new page or column that the table continues onto.

For the **nroff** formatter, the **−e** option [**−E** for **mm**(1) {2.1}] may be used for terminals, such as the 450, that are capable of finer printing resolution. This will cause better alignment of features such as the lines forming the corner of a box. The **−e** is not effective with **col**(1).

7.4 Equations

```
.DS
.EQ [label]
equation(s)
.EN
.DE
```

Mathematical typesetting programs **eqn/neqn**(1) expect to use the **.EQ** (equation start) and **.EN** (equation end) macros as delimiters in the same way that **tbl**(1) uses **.TS** and **.TE**; however, **.EQ** and **.EN** must

occur inside a **.DS/.DE** pair. There is an exception to this rule − if **.EQ** and **.EN** are used to specify only the delimiters for in-line equations or to specify **eqn/neqn** defines, the **.DS** and **.DE** macros must not be used; otherwise, extra blank lines will appear in the output.

The **.EQ** macro takes an argument that will be used as a label for the equation. By default, the label will appear at the right margin in the "vertical center" of the general equation. The **Eq** register may be set to 1 to change labeling to the left margin.

The equation will be centered for centered displays; otherwise, the equation will be adjusted to the opposite margin from the label.

7.5 Figure, Table, Equation, and Exhibit Titles

```
.FG [title] [override] [flag]
.TB [title] [override] [flag]
.EC [title] [override] [flag]
.EX [title] [override] [flag]
```

The **.FG** (figure title), **.TB** (table title), **.EC** (equation caption), and **.EX** (exhibit caption) macros are normally used inside **.DS/.DE** pairs to automatically number and title figures, tables, and equations. These macros use registers **Fg**, **Tb**, **Ec**, and **Ex**, respectively (see section 2.4 on **−rN5** to reset counters in sections). For example:

```
.FG "This is a Figure Title"
```

yields

Figure 1. This is a Figure Title

The **.TB** macro replaces "Figure" with "TABLE", the **.EC** macro replaces "Figure" with "Equation", and the **.EX** macro replaces "Figure" with "Exhibit". The output title is centered if it can fit on a single line; otherwise, all lines but the first are indented to line up with the first character of the title. The format of the numbers may be changed using the **.af** request of the formatter. By setting the **Of** register to 1, the format of the caption may be changed from

Figure 1. Title

to

Figure 1 – Title

The *override* argument is used to modify normal numbering. If the *flag* argument is omitted or 0, *override* is used as a prefix to the number; if the *flag* argument is 1, *override* is used as a suffix; and if the *flag* argument is 2, *override* replaces the number. If **–rN5** {2.4} is given, "section-figure" numbering is set automatically and user-specified *override* argument is ignored.

As a matter of formatting style, table headings are usually placed above the text of tables, while figure, equation, and exhibit titles are usually placed below corresponding figures and equations.

7.6 List of Figures, Tables, Equations, and Exhibits

A list of figures, tables, exhibits, and equations are printed following the table of contents if the number registers **Lf**, **Lt**, **Lx**, and **Le** (respectively) are set to 1. The **Lf**, **Lt**, and **Lx** registers are 1 by default; **Le** is 0 by default.

Titles of these lists may be changed by redefining the following strings which are shown here with their default values:

```
.ds Lf LIST OF FIGURES
.ds Lt LIST OF TABLES
.ds Lx LIST OF EXHIBITS
.ds Le LIST OF EQUATIONS
```

8. Footnotes

There are two macros (**.FS** and **.FE**) that delimit text of footnotes, a string (**F**) that automatically numbers footnotes, and a macro (**.FD**) that specifies the style of footnote text. Footnotes are processed in an environment different from that of the body of text (refer to **.ev** request in Chapter 3).

8.1 Automatic Numbering of Footnotes

Footnotes may be automatically numbered by typing the three characters "***F**" (i.e., invoking the string **F**) immediately after the text to be footnoted without any intervening spaces. This will place the next sequential footnote number (in a smaller point size) a half line above the text to be footnoted.

8.2 Delimiting Footnote Text

```
.FS [label]
one or more lines of footnote text
.FE
```

There are two macros that delimit the text of each footnote. The **.FS** (footnote start) macro marks the beginning of footnote text, and the **.FE** (footnote end) macro marks the end. The *label* on the **.FS** macro, if present, will be used to mark footnote text. Otherwise, the number retrieved from the string **F** will be used. Automatically numbered and user-labeled footnotes may be intermixed. If a footnote is labeled (**.FS** *label*), the text to be footnoted must be followed by *label*, rather than by "***F**". Text between **.FS** and **.FE** is processed in fill mode. Another **.FS**, a **.DS**, or a **.DF** are not permitted between **.FS** and **.FE** macros. If footnotes are required in the title, abstract, or table {7.3}, only labeled footnotes will appear properly. Everywhere else automatically numbered footnotes work correctly. For example, the input for an automatically numbered footnote is:

```
This is the line containing the word\*F
.FS
This is the text of the footnote.
.FE
to be footnoted and automatically numbered.
```

and the input for labeled footnote is:

```
This is a labeled*
.FS *
The footnote is labeled with an asterisk.
.FE
footnote.
```

Text of the footnote (enclosed within the **.FS**/**.FE** pair) should immediately follow the word to be footnoted in the input text, so that "***F**" or *label* occurs at the end of a line of input and the next line is the **.FS** macro call. It is also good practice to append an unpaddable space {3.3} to "***F**" or *label* when they follow an end-of-sentence punctuation mark (i.e., period, question mark, exclamation point).

8.3 Format Style of Footnote Text

.FD [*arg*] [1]

Within footnote text, the user can control formatting style by specifying text hyphenation, right margin justification, and text indentation, as well as left or right justification of the label when text indenting is used. The **.FD** macro is invoked to select the appropriate style.

The first argument (**arg**) is a number from the left column of the following table. Formatting style for each number is indicated in the remaining four columns. Further explanation of the first two of these columns is given in the definitions of the **.ad**, **.na**, **.hy**, and **.nh** (adjust, no adjust, hyphenation, and no hyphenation, respectively) requests in Chapter 3.

arg	HYPHENATION	ADJUST	TEXT INDENT	LABEL JUSTIFICATION
0	**.nh**	**.ad**	yes	left
1	**.hy**	**.ad**	yes	left
2	**.nh**	**.na**	yes	left
3	**.hy**	**.na**	yes	left
4	**.nh**	**.ad**	no	left
5	**.hy**	**.ad**	no	left
6	**.nh**	**.na**	no	left
7	**.hy**	**.na**	no	left
8	**.nh**	**.ad**	yes	right
9	**.hy**	**.ad**	yes	right
10	**.nh**	**.na**	yes	right
11	**.hy**	**.na**	yes	right

If the first argument to **.FD** is greater than 11, the effect is as if **.FD 0** were specified. If the first argument is omitted or null, the effect is equivalent to **.FD 10** in the **nroff** formatter and to **.FD 0** in the **troff** formatter; these are also the respective initial default values.

If the second argument is specified, then when a first-level heading is encountered, automatically numbered footnotes begin again with 1. This is most useful with the "section-page" page numbering scheme. As an example, the input line

.FD "" 1

maintains the default formatting style and causes footnotes to be numbered afresh after each first-level heading in a document.

Hyphenation across pages is inhibited by **mm** except for long footnotes that continue to the following page. If hyphenation is permitted, it is possible for the last word on the last line on the current page footnote to be hyphenated. The user has control over this situation by specifying an even **.FD** argument.

Footnotes are separated from the body of the text by a short line rule. Those that continue to the next page are separated from the body of the text by a full-width rule. In the **troff** formatter, footnotes are set in type two points smaller than the point size used in the body of text.

8.4 Spacing Between Footnote Entries

Normally, one blank line (**nroff**) or a 3-point vertical space (**troff**) separates footnotes when more than one occurs on a page. To change this spacing, the **Fs** number register is set to the desired value. For example:

.nr Fs 2

will cause two blank lines (**nroff**) a 6-point vertical space (**troff**) to occur between footnotes.

9. Page Headers and Footers

Text printed at the top of each page is called *page header*. Text printed at the bottom of each page is called *page footer*. There can be up to three lines of text associated with the header — every page, even page only, and odd page only. Thus the page header may have up to two lines of text — the line that occurs at the top of every page and the line for the even- or odd-numbered page. The same is true for the page footer.

This part describes the default appearance of page headers and page footers and ways of changing them. The term *header* (not qualified by *even* or *odd*) is used to mean the page header line that occurs on every page, and similarly for the term *footer*.

9.1 Default Headers and Footers

By default, each page has a centered page number as the header. There is no default footer and no even/odd default headers or footers except as specified in section 9.3.

In a memorandum or a released-paper style document, the page header on the first page is automatically suppressed provided a break does not occur before the **.MT** macro is called. Macros and text in the following categories do not cause a break and are permitted before the memorandum types (**.MT**) macro:

- Memorandum and released-paper style document macros (**.TL**, **.AU**, **.AT**, **.TM**, **.AS**, **.AE**, **.OK**, **.ND**, **.AF**, **.NS**, and **.NE**)
- Page headers and footers macros (**.PH**, **.EH**, **.OH**, **.PF**, **.EF**, and **.OF**)
- The **.nr** and **.ds** requests.

9.2 Header and Footer Macros

For header and footer macros (**.PH**, **.EH**, **.OH**, **.PF**, **.EF**, and **.OF**) the argument [*arg*] is of the form:

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

If it is inconvenient to use apostrophe (') as the delimiter because it occurs within one of the parts, it may be replaced uniformly by any other character. The **.fc** request redefines the delimiter. In formatted output, the parts are left justified, centered, and right justified, respectively.

9.2.1 Page Header

.PH [*arg*]

The **.PH** macro specifies the header that is to appear at the top of every page. The initial value is the default centered page number enclosed by hyphens. The page number contained in the **P** register is an Arabic number. The format of the number may be changed by the **.af** macro request.

If "*debug mode*" is set using the flag **−rD1** on the command line {2.4}, additional information printed at the top left of each page is included in

the default header. This consists of the Source Code Control System (SCCS) release and level of Memorandum Macros (thus identifying the current version {12.3}) followed by the current line number within the current input file.

9.2.2 Even-Page Header

.EH [*arg*]

The **.EH** macro supplies a line to be printed at the top of each even-numbered page immediately following the header. Initial value is a blank line.

9.2.3 Odd-Page Header

.OH [*arg*]

The **.OH** macro is the same as the **.EH** except that it applies to odd-numbered pages.

9.2.4 Page Footer

.PF [*arg*]

The **.PF** macro specifies the line that is to appear at the bottom of each page. Its initial value is a blank line. If the **−rC***n* flag is specified on the command line {2.4}, the type of copy follows the footer on a separate line. In particular, if **−rC3** or **−rC4** (DRAFT) is specified, the footer is initialized to contain the date {6.8} instead of being a blank line.

9.2.5 Even-Page Footer

.EF [*arg*]

The **.EF** macro supplies a line to be printed at the bottom of each even-numbered page immediately preceding the footer. Initial value is a blank line.

9.2.6 Odd-Page Footer

.OF [*arg*]

The **.OF** macro supplies a line to be printed at the bottom of each odd-numbered page immediately preceding the footer. Initial value is a blank line.

9.2.7 First Page Footer

By default, the first page footer is a blank line. If, in the input text file, the user specifies **.PF** and/or **.OF** before the end of the first page of the document, these lines will appear at the bottom of the first page.

The header (whatever its contents) replaces the footer on the first page only if the **−rN1** flag is specified on the command line {2.4}.

9.3 Default Header and Footer With Section-Page Numbering

Pages can be numbered sequentially within sections by "section-number page-number" {4.5}. To obtain this numbering style, **−rN3** or **−rN5** is specified on the command line. In this case, the default *footer* is a centered "section-page" number, e.g., 7-2; and the default page header is blank.

9.4 Strings and Registers in Header and Footer Macros

String and register names may be placed in arguments to header and footer macros. If the value of the string or register is to be computed when the respective header or footer is printed, invocation must be escaped by four backslashes. This is because string or register invocation will be processed three times:

1. As the argument to the header or footer macro
2. In a formatting request within the header or footer macro
3. In a **.tl** request during header or footer processing.

For example, page number register **P** must be escaped with four backslashes in order to specify a header in which the page number is to be printed at the right margin, e.g.:

```
.PH "''Page \\\\nP'"
```

creates a right-justified header containing the word "Page" followed by the page number. Similarly, to specify a footer with the "section-page" style, the user specifies (see section 4.2.2.5 for meaning of **H1**):


```
.PF "''− \\\\n(H1−\\\\nP −'"
```

If the user arranges for the string "**a]**" to contain the current section heading which is to be printed at the bottom of each page, the **.PF** macro call would be:

```
.PF "''\\\\*(a]''"
```

If only one or two backslashes were used, the footer would print a constant value for **a]**, namely, its value when **.PF** appeared in the input text.

9.5 Header and Footer Example

The following sequence specifies blank lines for header and footer lines, page numbers on the outside margin of each page (i.e., top left margin of even pages and top right margin of odd pages), and "Revision 3" on the top inside margin of each page (nothing is specified for the center):

```
.PH ""
.PF ""
.EH "'\\\\nP''Revision 3'"
.OH "'Revision 3''\\\\nP'"
```

9.6 Generalized Top-of-Page Processing

Note: This part is intended only for users accustomed to writing formatter macros.

During header processing, **mm** invokes two user-definable macros:

- The **.TP** (top of page) macro is invoked in the environment (refer to **.ev** request) of the header.
- The **.PX** is a page header user-exit macro that is invoked (without arguments) when the normal environment has been restored and with the "no-space" mode already in effect.

The effective initial definition of **.TP** (after the first page of a document) is

```
.de TP
.sp 3
.tl \\*()t
.if e 'tl \\*()e
.if o 'tl \\*()o
.sp 2
..
```

The string "}t" contains the header, the string "}e" contains the even-page header, and the string "}o" contains the odd-page header as defined by the **.PH**, **.EH**, and **.OH** macros, respectively. To obtain more specialized page titles, the user may redefine the **.TP** macro to cause the desired header processing {12.5}. Formatting done within the **.TP** macro is processed in an environment different from that of the body. For example, to obtain a page header that includes three centered lines of data, i.e., document number, issue date, and revision date, the user could define the **.TP** as follows:

```
.de TP
.sp
.ce 3
777-888-999
Iss. 2, AUG 1977
Rev. 7, SEP 1977
.sp
..
```

The **.PX** macro may be used to provide text that is to appear at the top of each page after the normal header and that may have tab stops to align it with columns of text in the body of the document.

9.7 Generalized Bottom-of-Page Processing

```
.BS
zero or more lines of text
.BE
```

Lines of text that are specified between the **.BS** (bottom-block start) and **.BE** (bottom-block end) macros will be printed at the bottom of each page after the footnotes (if any) but before the page footer. This block of text is removed by specifying an empty block, i.e.:


```
.BS
.BE
```

The bottom block will appear on the table of contents, pages, and the cover sheet for memorandum for file, but not on the technical memorandum or released-paper cover sheets.

9.8 Top and Bottom (Vertical) Margins

.VM [*top*] [*bottom*]

The **.VM** (vertical margin) macro allows the user to specify additional space at the top and bottom of the page. This space precedes the page header and follows the page footer. The **.VM** macro takes two unscaled arguments that are treated as vertical spaces (**v**). For example:

```
.VM 10 15
```

adds 10 vertical spaces to the default top of page margin and 15 vertical spaces to the default bottom of page margin. Both arguments must be positive (default spacing at the top of the page may be decreased by redefining **.TP**).

9.9 Proprietary Marking

.PM [*code*]

The **.PM** (proprietary marking) macro appends to the page footer a PRIVATE, NOTICE, BELL LABORATORIES PROPRIETARY, or BELL LABORATORIES RESTRICTED disclaimer. The *code* argument may be:

CODE	DISCLAIMER
none	turn off previous disclaimer, if any
P	PRIVATE
N	NOTICE
BP	BELL LABORATORIES PROPRIETARY
BR	BELL LABORATORIES RESTRICTED

These disclaimers are in a form approved for use by the Bell System. The user may alternate disclaimers by use of the **.BS**/**.BE** macro pair.

9.10 Private Documents

.nr Pv *value*

The word "PRIVATE" may be printed, centered, and underlined on the second line of a document (preceding the page header). This is done by setting the **Pv** register *value*:

VALUE	MEANING
0	do not print PRIVATE (default)
1	PRIVATE on first page only
2	PRIVATE on all pages

If *value* is 2, the user definable **.TP** macro may not be used because the **.TP** macro is used by **mm** to print "PRIVATE" on all pages except the first page of a memorandum on which **.TP** is not invoked.

10. Table of Contents and Cover Sheet

The table of contents and the cover sheet for a document are produced by invoking the **.TC** and **.CS** macros, respectively.

Note: This section refers to cover sheets for technical memoranda and released papers only. The mechanism for producing a memorandum for file cover sheet was discussed earlier {6.5}.

These macros normally appear once at the end of the document, after the Signature Block {6.11.1} and Notations {6.11.2} macros, and may occur in either order.

The table of contents is produced at the end of the document because the entire document must be processed before the table of contents can be generated. Similarly, the cover sheet may not be desired by a user and is therefore produced at the end.

10.1 Table of Contents

.TC [*slevel*] [*spacing*] [*tlevel*] [*tab*] [*head1*] [*head2*] [*head3*] [*head4*] [*head5*]

The **.TC** macro generates a table of contents containing heading levels that were saved for the table of contents as determined by the value of the **Cl** register {4.4}. Arguments to **.TC** control spacing before each entry, placement of associated page number, and additional text on the first page of the table of contents before the word "CONTENTS".

Spacing before each entry is controlled by the first and second arguments (*slevel* and *spacing*). Headings whose level is less than or equal to *slevel* will have *spacing* blank lines (**nroff**) or half vertical spaces (**troff**) before them. Both *slevel* and *spacing* default to 1. This means that first-level headings are preceded by one blank line (**nroff**) or one-half a vertical space (**troff**). The *slevel* argument does not control what levels of heading have been saved; saving of headings is the function of the **Cl** register.

The third and fourth arguments (*tlevel* and *tab*) control placement of associated page number for each heading. Page numbers can be justified at the right margin with either blanks or dots (called leaders) separating the heading text from the page number, or the page numbers can follow the heading text.

For headings whose level is less than or equal to *tlevel* (default 2), page numbers are justified at the right margin. In this case, the value of *tab* determines the character used to separate heading text from page number. If *tab* is 0 (default value), dots (i.e., leaders) are used. If *tab* is greater than 0, spaces are used.

For headings whose level is greater than *tlevel*, page numbers are separated from heading text by two spaces (i.e., page numbers are "ragged right", not right justified).

Additional arguments (*head1* ... *head5*) are horizontally centered on the page and precede the table of contents.

If the **.TC** macro is invoked with at most four arguments, the user-exit macro **.TX** is invoked (without arguments) before the word "CONTENTS" is printed or the user-exit macro **.TY** is invoked and the word "CONTENTS" is not printed.

By defining **.TX** or **.TY** and invoking **.TC** with at most four arguments, the user can specify what needs to be done at the top of the first page

of the table of contents. For example:

```
.de TX
.ce 2
Special Application
Message Transmission
.sp
.in +10n
Approved: \l'3i'
.in 0
.sp
..
.TC
```

yields the following output when the file is formatted

Special Application
Message Transmission

Approved: _____

CONTENTS

.
.
.

If the **.TX** macro were defined as **.TY**, the word "CONTENTS" would be suppressed. Defining **.TY** as an empty macro will suppress "CONTENTS" with no replacement:

```
.de TY
..
```

By default, the first level headings will appear in the table of contents left justified. Subsequent levels will be aligned with the text of headings at the preceding level. These indentations may be changed by defining the **Ci** string which takes a maximum of seven arguments corresponding to the heading levels. It must be given at least as many arguments as are set by the **Cl** register. Arguments must be scaled. For example, with "**Cl** = 5":

```
.ds Ci .25i .5i .75i 1i 1i  \"troff
```

or


```
.ds Ci 0 2n 4n 6n 8n  \"nroff
```

Two other registers are available to modify the format of the table of contents – **Oc** and **Cp**.

By default, table of contents pages will have lowercase Roman numeral page numbering. If the **Oc** register is set to 1, the **.TC** macro will not print any page number but will instead reset the **P** register to 1. It is the user's responsibility to give an appropriate page footer to specify the placement of the page number. Ordinarily, the same **.PF** macro (page footer) used in the body of the document will be adequate.

The list of figures, tables, etc. pages will be produced separately unless **Cp** is set to 1 which causes these lists to appear on the same page as the table of contents.

10.2 Cover Sheet

.CS [*pages*] [*other*] [*total*] [*figs*] [*tbls*] [*refs*]

The **.CS** macro generates a cover sheet in either the released paper or technical memorandum style (see section 6.5 for details of the memorandum for file cover sheet). All other information for the cover sheet is obtained from data given before the **.MT** macro call {6.1}. If the technical memorandum style is used, the **.CS** macro generates the "Cover Sheet for Technical Memorandum". The data that appear in the lower left corner of the technical memorandum cover sheet (counts of: pages of text, other pages, total pages, figures, tables, and references) are generated automatically (0 is used for "other pages"). These values may be changed by supplying the corresponding arguments to the **.CS** macro. If the released-paper style is used, all arguments to **.CS** are ignored.

11. References

There are two macros (**.RS** and **.RF**) that delimit the text of references, a string that automatically numbers the subsequent references, and an optional macro (**.RP**) that produces reference pages within the document.

11.1 Automatic Numbering of References

Automatically numbered references may be obtained by typing *(Rf (invoking the string **Rf**) immediately after the text to be referenced. This places the next sequential reference number (in a smaller point size) enclosed in brackets one-half line above the text to be referenced. Reference count is kept in the **Rf** number register.

11.2 Delimiting Reference Text

.RS [*string-name*]
.RF

The **.RS** and **.RF** macros are used to delimit text of each reference as shown below:

```
A line of text to be referenced.\*(Rf
.RS
reference text
.RF
```

11.3 Subsequent References

The **.RS** macro takes one argument, a *string-name*. For example:

```
.RS aA
reference text
.RF
```

The string **aA** is assigned the current reference number. This string may be used later in the document as the string call, ***(aA**, to reference text which must be labeled with a prior reference number. The reference is output enclosed in brackets one-half line above the text to be referenced. No **.RS/.RF** pair is needed for subsequent references.

11.4 Reference Page

.RP [*arg1*] [*arg2*]

A reference page, entitled by default "References", will be generated automatically at the end of the document (before table of contents and cover sheet) and will be listed in the table of contents. This page contains the reference items (i.e., reference text enclosed within **.RS/.RF** pairs). Reference items will be separated by a space (**nroff**) or one-half

a vertical space (**troff**) unless the **Ls** register is set to 0 to suppress this spacing. The user may change the reference page title by defining the **Rp** string:

```
.ds Rp "New Title"
```

The **.RP** (reference page) macro may be used to produce reference pages anywhere else within a document (i.e., after each major section). It is not needed to produce a separate reference page with default spacings at the end of the document.

Two **.RP** macro arguments allow the user to control resetting of reference numbering and page skipping.

arg1	MEANING
0	reset reference counter (default)
1	do not reset reference counter

arg2	MEANING
0	put on separate page (default)
1	do not cause a following **.SK**
2	do not cause a preceding **.SK**
3	no **.SK** before or after

If no **.SK** macro is issued by the **.RP** macro, a single blank line will separate the references from the following/preceding text. The user may wish to adjust spacing. For example, to produce references at the end of each major section:

```
.sp 3
.RP 1 2
.H 1 "Next Section"
```

12. Miscellaneous Features

12.1 Bold, Italic, and Roman Fonts

.B [*bold-arg*] [*previous-font-arg*] ...
.I [*italic-arg*] [*previous-font-arg*] ...
.R

When called without arguments, the **.B** macro changes the font to bold and the **.I** macro changes to underlining (**nroff**) or italic (**troff**). This condition continues until the occurrence of the **.R** macro which causes the Roman font to be restored. Thus:

```
.I
here is some text.
.R
```

yields underlined text via **nroff**(1) and italic text via **troff**(1).

If the **.B** or **.I** macro is called with one argument, that argument is printed in the appropriate font (underlined in the **nroff** formatter for **.I**). Then the previous font is restored (underlining is turned off in the **nroff** formatter). If two or more arguments (maximum six) are given with a **.B** or **.I** macro call, the second argument is concatenated to the first with no intervening space (1/12 space if the first font is italic) but is printed in the previous font. Remaining pairs of arguments are similarly alternated. For example:

```
.I one " two " three -four
```

produces

one two *three*-four

The **.B** and **.I** macros alternate with the prevailing font at the time the macros are invoked. To alternate specific pairs of fonts, the following macros are available:

.IB – italic bold
.BI – bold italic
.IR – italic Roman
.RI – Roman italic
.RB – Roman bold
.BR – bold Roman

Each macro takes a maximum of six arguments and alternates arguments between specified fonts.

When using a terminal that cannot underline, the following can be inserted at the beginning of the document to eliminate all underlining:


```
.rm ul
.rm cu
```

Note: Font changes in headings are handled separately {4.2.2.4.1}.

12.2 Justification of Right Margin

.SA [*arg*]

The **.SA** macro is used to set right-margin justification for the main body of text. Two justification flags are used – *current* and *default*. Initially, both flags are set for no justification in the **nroff** formatter and for justification in the **troff** formatter. The argument causes the following action:

arg	MEANING
0	Sets both flags to no justification. It acts like the **.na** request.
1	Sets both flags to cause both right and left justification, the same as the **.ad** request.
omitted	Causes the current flag to be copied from the default flag, thus performing either a **.na** or **.ad** depending on the default condition.

In general, the no adjust request (**.na**) can be used to ensure that justification is turned off, but **.SA** should be used to restore justification, rather than the **.ad** request. In this way, justification or no justification for the remainder of the text is specified by inserting "**.SA 0**" or "**.SA 1**" once at the beginning of the document.

12.3 SCCS Release Identification

The **RE** string contains the SCCS release and the Memorandum Macros text formatting package current version level. For example:

```
This is version \*(RE of the macros.
```

produces

This is version 10.129 of the macros.

This information is useful in analyzing suspected bugs in **mm**. The easiest way to have the release identification number appear in the output is to specify **−rD1** {2.4} on the command line. This causes the **RE** string to be output as part of the page header {9.2.1}.

12.4 Two-Column Output

.2C
text and formatting requests (except another **.2C**)
.1C

The Memorandum Macros text formatting package can format two-columns on a page. The **.2C** macro begins 2-column processing which continues until a **.1C** macro (1-column processing) is encountered. In 2-column processing, each physical page is thought of as containing 2-columnar "pages" of equal (but smaller) "page" width. Page headers and footers are not affected by 2-column processing. The **.2C** macro does not balance 2-column output.

It is possible to have full-page width footnotes and displays when in 2-column mode, although default action is for footnotes and displays to be narrow in 2-column mode and wide in 1-column mode. Footnote and display width is controlled by the **.WC** (width control) macro, which takes the following arguments:

arg	MEANING
N	Default mode (**−WF**, **−FF**, **−WD**, **FB**).
WF	Wide footnotes (even in 2-column mode).
−WF	DEFAULT: Turn off **WF**. Footnotes follow column mode; wide in 1-column mode (**1C**), narrow in 2-column mode (**2C**), unless **FF** is set.
FF	First footnote. All footnotes have same width as first footnote encountered for that page.
−FF	DEFAULT: Turn off **FF**. Footnote style follows settings of **WF** or **−WF**.
WD	Wide displays (even in 2-column mode).
−WD	DEFAULT: Displays follow the column mode in effect when display is encountered.

FB	DEFAULT: Floating displays cause a break when output on the current page.
−FB	Floating displays on current page do not cause a break.

Note: The **.WC WD FF** command will cause all displays to be wide and all footnotes on a page to be the same width while **.WC N** will reinstate default actions. If conflicting settings are given to **.WC**, the last one is used. A **.WC WF −WF** command has the effect of a **.WC −WF**.

12.5 Column Headings for Two-Column Output

Note: This section is intended only for users accustomed to writing formatter macros.

In 2-column processing output, it is sometimes necessary to have headers over each column, as well as headers over the entire page. This is accomplished by redefining the **.TP** macro {9.6} to provide header lines both for the entire page and for each of the columns. For example:

```
.de TP
.sp 2
.tl 'Page \\nP'OVERALL"
.tl "TITLE"
.sp
.nf
.ta 16C 31R 34 50C 65R
left→ center→ right→ left→ center→ right
→ first column→ → → second column
.fi
.sp 2
..
```

where → stands for the tab character.

The above example will produce two lines of page header text plus two lines of headers over each column. Tab stops are for a 65-en overall line length.

12.6 Vertical Spacing

.SP [*lines*]

There exists several ways of obtaining vertical spacing, all with different effects. The **.sp** request spaces the number of lines specified unless the no space (**.ns**) mode is on, then the **.sp** request is ignored. The no space mode is set at the end of a page header to eliminate spacing by a **.sp** or **.bp** request that happens to occur at the top of a page. This mode can be turned off by the **.rs** (restore spacing) request.

The **.SP** macro is used to avoid the accumulation of vertical space by successive macro calls. Several **.SP** calls in a row will not produce the sum of the arguments but only the maximum argument. For example, the following produces only three blank lines:

```
.SP 2
.SP 3
.SP
```

Many Memorandum Macros utilize **.SP** for spacing. For example, "**.LE 1**" {5.9} immediately followed by "**.P**" {4.1} produces only a single blank line (**nroff**) or one-half a vertical space (**troff**) between the end of the list and the following paragraph. An omitted argument defaults to one blank line (**nroff**) or one vertical space (**troff**). Negative arguments are not permitted. The argument must be unscaled but fractional amounts are permitted. The **.SP** macro (as well as **.sp**) is also inhibited by the **.ns** (no space) request.

12.7 Skipping Pages

.SK [*pages*]

The **.SK** macro skips pages but retains the usual header and footer processing. If the *pages* argument is omitted, null, or 0, **.SK** skips to the top of the next page unless it is currently at the top of a page (then it does nothing). A "**.SK** *n*" command skips *n* pages. A "**.SK**" positions text that follows it at the top of a page, while "**.SK 1**" leaves one page blank except for the header and footer.

12.8 Forcing an Odd Page

.OP

The **.OP** macro is used to ensure that formatted output text following the macro begins at the top of an odd-numbered page.

- If currently at the top of an odd-numbered page, text output begins on that page (no motion takes place).
- If currently on an even page, text resumes printing at the top of the next page.
- If currently on an odd page (but not at the top of the page), one blank page is produced, and printing resumes on the next odd-numbered page after that.

12.9 Setting Point Size and Vertical Spacing

.S [*point size*] [*vertical spacing*]

The prevailing point size and vertical spacing may be changed by invoking the **.S** macro. In the **troff** formatter, the default point size (obtained from the **mm** register **S** {2.4}) is 10 points, and the vertical spacing is 12 points (six lines per inch). The mnemonics **D** (default value), **C** (current value), and **P** (previous value) may be used for both arguments.

- If an argument is *negative*, current value is decremented by the specified amount.
- If an argument is *positive*, current value is incremented by the specified amount.
- If an argument is *unsigned*, it is used as the new value.
- If there are no arguments, the **.S** macro defaults to **P**.
- If the first argument is specified but the second is not, then (default) **D** is used for the vertical spacing.

Default value for vertical spacing is always two points greater than the current point size. Footnotes {8} are two points smaller than the body with an additional 3-point space between footnotes. A null ("") value for either argument defaults to **C** (current value). Thus, if *n* is a numeric value:

```
.S        =   .S P P
.S "" n   =   .S C n
.S n ""   =   .S n C
.S n      =   .S n D
.S ""     =   .S C D
.S "" ""  =   .S C C
.S n n    =   .S n n
```

If the first argument is greater than 99, the default point size (10 points) is restored. If the second argument is greater than 99, the default vertical spacing (current point size plus two points) is used. For example:

```
.S 100      =   .S 10 12
.S 14 111   =   .S 14 16
```

12.10 Reducing Point Size of a String

.SM *string1* [*string2*] [*string3*]

The **.SM** macro allows the user to reduce by one point the size of a string. If the third argument (*string3*) is omitted, the first argument (*string1*) is made smaller and is concatenated with the second argument (*string2*) if specified. If all three arguments are present (even if any are null), the second argument is made smaller and all three arguments are concatenated. For example:

INPUT	OUTPUT
.SM X	X
.SM X Y	XY
.SM Y X Y	YXY
.SM YXYX	YXYX
.SM YXYX)	YXYX)
.SM (YXYX)	(YXYX)
.SM Y XYX ""	YXYX

12.11 Producing Accents

Strings may be used to produce accents for letters as shown in the following examples:

	INPUT	OUTPUT
Grave accent	c*‘	c̀
Acute accent	e*’	é
Circumflex	o*^	ô
Tilde	n*~	ñ
Cedilla	c*,	ç
Lower-case umlaut	u*:	ü
Upper-case umlaut	U*;	Ü

12.12 Inserting Text Interactively

.RD [*prompt*] [*diversion*] [*string*]

The **.RD** (read insertion) macro allows a user to stop the standard output of a document and to read text from the standard input until two consecutive newline characters are found. When newline characters are encountered, normal output is resumed.

- The *prompt* argument will be printed at the terminal. If not given, **.RD** signals the user with a BEL on terminal output.
- The *diversion* argument allows the user to save all text typed in after the prompt in a macro whose name is that of the diversion.
- The *string* argument allows the user to save for later reference the first line following the prompt in the named string.

The **.RD** macro follows the formatting conventions in effect. Thus, the following examples assume that the **.RD** is invoked in no fill mode (**.nf**):

```
.RD Name aA bB
```

produces

```
Name: S. Jones   (user types name)
16 Elm Rd.,
Piscataway
```

The diverted macro **.aA** will contain

S. Jones
16 Elm Rd.,
Piscataway

The string **bB** (*(**bB**) contains "S. Jones".

A newline character followed by an EOF (user specifiable CONTROL-d) also allows the user to resume normal output.

13. Errors and Debugging

13.1 Error Terminations

When a macro detects an error, the following actions occur:

- A break occurs.
- The formatter output buffer (which may contain some text) is printed to avoid confusion regarding location of the error.
- A short message is printed giving the name of the macro that detected the error, type of error, and approximate line number in the current input file of the last processed input line. Error messages are explained in Table 16.D.
- Processing terminates unless register **D** {2.4} has a positive value. In the latter case, processing continues even though the output is guaranteed to be deranged from that point on.

The error message is printed by outputting the message directly to the user terminal. If an output filter, such as **300**(1), **450**(1), or **hp**(1) is being used to post-process the **nroff** formatter output, the message may be garbled by being intermixed with text held in that filter's output buffer.

Note: If any of **cw**(1), **eqn/neqn**(1), and **tbl**(1) programs are being used and if the **-o***list* option of the formatter causes the last page of the document not to be printed, a harmless "broken pipe" message may result.

13.2 Disappearance of Output

Disappearance of output usually occurs because of an unclosed diversion (e.g., a missing **.DE** or **.FE** macro). Fortunately, macros that use

diversions are careful about it, and these macros check to make sure that illegal nestings do not occur. If any error message is issued concerning a missing **.DE** or **.FE**, the appropriate action is to search backwards from the termination point looking for the corresponding associated **.DF**, **.DS**, or **.FS** (since these macros are used in pairs).

The following command:

grep −n '^\.[EDFRT][EFNQS]' filename1 filename2

prints all the **.DF**, **.DS**, **.DE**, **.EQ**, **.EN**, **.FS**, **.FE**, **.RS**, **.RF**, **.TS**, and **.TE** macros found in *filename1* and *filename2* each preceded by its file name and the line number in that file. This listing can be used to check for illegal nesting and/or omission of these macros.

14. Extending and Modifying Memorandum Macros

14.1 Naming Conventions

In this part, the following conventions are used to describe names:

n:	Digit
a:	Lowercase letter
A:	Uppercase letter
x:	Any alphanumeric character (n, a, or A, i.e., letter or digit)
s:	Any nonalphanumeric character (special character)

All other characters are literals (characters that stand for themselves).

Request, macro, and string names are kept by the formatters in a single internal table; therefore, there must be no duplication among such names. Number register names are kept in a separate table.

14.1.1 Names Used by Formatters

requests:	*aa* (most common)
	an (only one, currently: **c2**)
registers:	*aa* (normal)
	.x (normal)
	.s (only one, currently: **.**)
	a. (only one, currently: **c.**)
	% (page number)

14.1.2 Names Used by Memorandum Macros

macros and strings:	*A*, *AA*, *Aa* (accessible to users; e.g., macros **P** and **HU**, strings **F**, **BU**, and **Lt**)
	nA (accessible to users; only two, currently: **1C** and **2C**)
	aA (accessible to users; only one, currently: **nP**)
	s (accessible to users; only the seven accents, currently {12.10})
	)*x*, }*x*,]*x*, >*x*, ?*x* (internal)
registers:	*An*, *Aa* (accessible to users; e.g., **H1**, **Fg**)
	A (accessible to users; meant to be set on the command line; e.g., **C**)
	:*x*, ;*x*, #*x*, ?*x*, !*x* (internal)

14.1.3 Names Used by cw, eqn/neqn, and tbl

The **cw**(1) program is the constant-width font preprocessor for the **troff** formatter. It uses the following five macro names:

.CD .CN .CP .CW .PC

This preprocessor also uses the number register names **cE** and **cW**. Mathematical equation preprocessors, **eqn**(1) and **neqn**(1), use registers and string names of the form **nn**. The table preprocessor, **tbl**(1), uses **T&**, **T#**, and **TW**, and names of the form:

a− a+ a| nn na ^a #a #s

14.1.4 Names Defined by User

Names that consist either of a single lowercase letter or a lowercase letter followed by a character other than a lowercase letter (names **.c2** and **.nP** are already used) should be used to avoid duplication with

already used names. The following is a possible naming convention:

macros:	*aA* (e.g., **bG**, **kW**)
strings:	*as* (e.g., **c)**, **f|**, **p|**)
registers:	*a* (e.g., **f**, **t**)

14.2 Sample Extensions

14.2.1 Appendix Headings

The following is a way of generating and numbering appendix headings:

```
.nr Hu 1
.nr a 0
.de aH
.nr a +1
.nr P 0
.PH "'''Appendix \\na-\\\\\\\\nP'"
.SK
.HU "\\1"
..
```

After the above initialization and definition, each call of the form

.aH "*title*"

begins a new page (with the page header changed to "Appendix *a−n*") and generates an unnumbered heading of *title*, which, if desired, can be saved for the table of contents. To center appendix titles the **Hc** register must be set to 1 {4.2.2.3}.

14.2.2 Hanging Indent With Tabs.

The following example illustrates the use of the hanging indent feature of variable-item lists {5.8}. A user-defined macro is defined to accept four arguments that make up the *mark*. In the output, each argument is to be separated from the previous one by a tab; tab settings are defined later. Since the first argument may begin with a period or apostrophe, the "\&" is used so that the formatter will not interpret such a line as a formatter request or macro call.

Note: The 2-character sequence "\&" is understood by formatters to be a "zero-width" space. It causes no output characters to appear, but it removes the special meaning of a leading period or apostrophe.

The "\t" is translated by the formatter into a tab. The "\c" is used to concatenate the input text that follows the macro call to the line built by the macro. The user-defined macro and an example of its use are:

```
.de aX
.LI
\&\\$1\t\\$2\t\\$3\t\\$4\t\c
..
.
.
.
.ta .5i 1i 1.5i 2i
.VL 2i
.aX .nh off \- no
No hyphenation.
Automatic hyphenation is turned off.
Words containing hyphens
(e.g., mother-in-law) may still be split across lines.
.aX .hy on \- no
Hyphenate.
Automatic hyphenation is turned on.
.aX .hc\<sp>c none none no
Hyphenation indicator character is set to "c" or removed.
During text processing, the indicator is suppressed
and will not appear in the output.
Prepending the indicator to a word has the effect
of preventing hyphenation of that word.
.LE
```

where <sp> stands for a space.

The resulting output is:

.nh	off	–	no	No hyphenation. Automatic hyphenation is turned off. Words containing hyphens (e.g., mother-in-law) may still be split across lines.
.hy	on	–	no	Hyphenate. Automatic hyphenation is turned on.
.hc c	none	none	no	Hyphenation indicator character is set to "c" or removed. During text processing, the indicator is suppressed and will not appear in the output. Prepending the indicator to a word has the effect of preventing hyphenation of that word.

15. Summary

The following are qualities of **mm** that have been emphasized in its design in approximate order of importance:

- *Robustness in the face of error* – A user need not be an **nroff/troff** expert to use the Memorandum Macros. When the input is incorrect, either the macros attempt to make a reasonable interpretation of the error or an error message describing the error is produced. An effort has been made to minimize the possibility that a user would get cryptic system messages or strange output as a result of simple errors.
- *Ease of use for simple documents* – It is not necessary to write complex sequences of commands to produce documents. Reasonable macro argument default values are provided where possible.
- *Parameterization* – There are many different preferences in the area of document styling. Many parameters are provided so that users can adapt input text files to produce output documents to their respective needs over a wide range of styles.
- *Extension by moderately expert users* – A strong effort has been made to use mnemonic naming conventions and consistent techniques in construction of macros. Naming conventions are given so that a user can add new macros or redefine existing ones if necessary.
- *Device independence* – A common use of **mm** is to produce documents on hard copy via teletypewriter terminals using the **nroff** formatter. Macros can be used conveniently with both 10- and 12-pitch terminals. In addition, output can be displayed on an appropriate CRT terminal. Macros have been constructed to allow compatibility with the **troff**(1) formatter so that output can be produced on both a phototypesetter and a teletypewriter/CRT terminal.
- *Minimization of input* – The design of macros attempts to minimize repetitive typing. For example, if a user wants to have a blank line after all first- or second-level headings, the user need only set a specific parameter once at the beginning of a document rather

than type a blank line after each such heading.

- *Decoupling of input format from output style* — There is but one way to prepare the input text although the user may obtain a number of output styles by setting a few global flags. For example, the .H macro is used for all numbered headings, yet the actual output style of these headings may be made to vary from document to document or within a single document.

16. Figures and Tables

This section contains Figure 16.A which is an example of an input file of a simple letter that is also shown formatted by both **nroff** and **troff** using the Memorandum Macros. This example illustrates how the formatters work and what to expect from your input file.

There are also four tables in this section that are useful reference tools when using the Memorandum Macros. The tables are:

Table 16.A — *Macro Names*: This table is a alphabetical summary of all the Memorandum Macro names available for producing a document.

Table 16.B — *String Names*: This table is a summary of all the predefined string names in the Memorandum Macro package.

Table 16.C — *Number Register Names*: This table is a summary of all the predefined number register names in the Memorandum Macro package.

Table 16.D — *Error Messages*: This table is a list of error messages that you may encounter when formatting a document. Memorandum Macro error messages as well as **nroff/troff** error messages are explained.

INPUT:

```
.nr N 2          \" specifies header to be omitted from page 1
.ta 3i
September 15, 1984
.SP 2
Mr. Steven J. Jones
.br
386 Broderick Street
.br
San Francisco, CA  94111
.SP
Dear Mr. Jones:
.P
Enclosed please find a copy of the
.I
UniPlus+\*F System V Document Processing Guide.
.R
.FS
UniPlus+ is a trademark of UniSoft Corporation.
.FE
.P
This manual is intended for use by those
who intend to use the \s-1UNIX\s+1\*F
.FS
\s-1UNIX\s+1 is a trademark of AT&T Bell Laboratories.
.FE
operating system for preparing documentation.
.P
This manual covers topics such as:
.VL 17
.LI \fIFormatters:\fR
the \fBnroff/troff\fR formatters are discussed in great detail
with tables listing defaults and explanations of all requests.
.LI \fITables:\fR
the \fBtbl\fR program is explained
with very helpful examples at the end of the chapter.
.LI \fIEquations:\fR
input of mathematical expressions is made simple by the \fBeqn\fR program;
the text contains many examples.
.LI "\fIMacro\ Package:\fR"
the \fBmm\fR macro package chapter gives a complete outline
of all the capabilities of this powerful document processing tool.
.LE
.P
I hope you will find this guide useful in preparing your report.
.SP
.nf
Sincerely,
.SP 2
Rosemary Clooney
Documentation Specialist
RC/jfb
Enc.
.fi
```

Figure 16.A. Example of a Simple Letter — Input File

nroff OUTPUT:

September 15, 1984

Mr. Steven J. Jones
386 Broderick Street
San Francisco, CA 94111

Dear Mr. Jones:

Enclosed please find a copy of the UniPlus+1 System V Document Processing Guide.

This manual is intended for use by those who intend to use the UNIX2 operating system for preparing documentation.

This manual covers topics such as:

Formatters: the **nroff/troff** formatters are discussed in great detail with tables listing defaults and explanations of all requests.

Tables: the **tbl** program is explained with very helpful examples at the end of the chapter.

Equations: input of mathematical expressions is made simple by the **eqn** program; the text contains many examples.

Macro Package: the **mm** macro package chapter gives a complete outline of all the capabilities of this powerful document processing tool.

I hope you will find this guide useful in preparing your report.

Sincerely,

Rosemary Clooney
Documentation Specialist

RC/jfb
Enc.

---

1. UniPlus+ is a trademark of UniSoft Corporation.
2. UNIX is a trademark of AT&T Bell Laboratories.

Figure 16.A. Example of a Simple Letter – NROFF Output (continued)

troff OUTPUT

September 15, 1984

Mr. Steven J. Jones
386 Broderick Street
San Francisco, CA 94111

Dear Mr. Jones:

Enclosed please find a copy of the *UniPlus+*[1] *System V Document Processing Guide.*

This manual is intended for use by those who intend to use the UNIX[2] operating system for preparing documentation.

This manual covers topics such as:

Formatters: the **nroff/troff** formatters are discussed in great detail with tables listing defaults and explanations of all requests.

Tables: the **tbl** program is explained with very helpful examples at the end of the chapter.

Equations: input of mathematical expressions is made simple by the **eqn** program; the text contains many examples.

Macro Package: the **mm** macro package chapter gives a complete outline of all the capabilities of this powerful document processing tool.

I hope you will find this guide useful in preparing your report.

Sincerely,

Rosemary Clooney
Documentation Specialist

RC/jfb
Enc.

---

1. UniPlus+ is a trademark of UniSoft Corporation.
2. UNIX is a trademark of AT&T Bell Laboratories.

Figure 16.A. Example of a Simple Letter – TROFF Output

TABLE 16.A. Memorandum Macro Names

MACRO	SECTION	DESCRIPTION
1C	12.4	1-column processing .1C
2C	12.4	2-column processing .2C
AE	6.5	Abstract end .AE
AF	6.9	Alternate format of "Subject/Date/From" block .AF [company-name]
AL	5.3	Automatically incremented list start .AL [type] [text-indent] [1]
AS	6.5	Abstract start .AS [arg] [indent]
AT	6.3	Author's title .AT [title] ...
AU	6.3	Author information .AU name [initials] [loc] [dept] [ext] [room] [arg] [arg] [arg]
AV	6.11.3	Approval signature .AV [name]
B	12.1	Bold .B [bold-arg] [prev-font-arg] [bold] [prev] [bold] [prev]
BE	9.7	Bottom block end .BE
BI	12.1	Bold/Italic .BI [bold-arg] [italic-arg] [bold] [italic] [bold] [italic]
BL	5.4	Bullet list start .BL [text-indent] [1]

* Macros marked with an asterisk are not, in general, called (invoked) directly by the user. They are "user exits" defined by the user and called by the Memorandum Macros from inside header, footer, or other macros.

Table 16.A. Memorandum Macro Names (continued)

MACRO	SECTION	DESCRIPTION
BR	12.1	Bold/Roman .BR [bold-arg] [Roman-arg] [bold] [Roman] [bold] [Roman]
BS	9.7	Bottom block start .BS
CS	10.2	Cover sheet .CS [pages] [other] [total] [figs] [tbls] [refs]
DE	7.1	Display end .DE
DF	7.2	Display floating start .DF [format] [fill] [right-indent]
DL	5.5	Dash list start .DL [text-indent] [1]
DS	7.1	Display static start .DS [format] [fill] [right-indent]
EC	7.5	Equation caption .EC [title] [override] [flag]
EF	9.2.5	Even-page footer .EF [arg]
EH	9.2.2	Even-page header .EH [arg]
EN	7.4	End equation display .EN
EQ	7.4	Equation display start .EQ [label]
EX	7.5	Exhibit caption .EX [title] [override] [flag]
FC	6.11.1	Formal closing .FC [closing]
FD	8.3	Footnote default format .FD [arg] [1]

Table 16.A. Memorandum Macro Names
(continued)

MACRO	SECTION	DESCRIPTION
FE	8.2	Footnote end .FE
FG	7.5	Figure title .FG [title] [override] [flag]
FS	8.2	Footnote start .FS [label]
H	4.2	Heading—numbered .H level [heading-text] [heading-suffix]
HC	3.4	Hyphenation character .HC [hyphenation-indicator]
HM	4.2.2.5	Heading mark style (Arabic or Roman numerals, or letters) .HM [arg1] ... [arg7]
HU	4.3	Heading—unnumbered .HU heading-text
HX*	4.6	Heading user exit X (before printing heading) .HX dlevel rlevel heading-text
HY*	4.6	Heading user exit Y (before printing heading) .HY dlevel rlevel heading-text
HZ*	4.6	Heading user exit Z (after printing heading) .HZ dlevel rlevel heading-text
I	12.1	Italic (underline in the **nroff** formatter) .I [italic-arg] [prev-font-arg] [italic] [prev] [italic] [prev]
IB	12.1	Italic/Bold .IB [italic-arg] [bold-arg] [italic] [bold] [italic] [bold]
IR	12.1	Italic/Roman .IR [italic-arg] [Roman-arg] [italic] [Roman] [italic] [Roman]
LB	5.12	List begin .LB text-indent mark-indent pad type [mark] [LI-space] [LB-space]

Table 16.A. Memorandum Macro Names
(continued)

MACRO	SECTION	DESCRIPTION
LC	5.13	List-status clear .LC [list-level]
LE	5.10	List end .LE [1]
LI	5.9	List item .LI [mark] [1]
ML	5.6	Marked list start .ML mark [text-indent] [1]
MT	6.7	Memorandum type .MT [type] [addressee] *or* .MT [4] [1]
ND	6.8	New date .ND new-date
NE	6.11.2	Notation end .NE
NS	6.11.2	Notation start .NS [arg]
nP	4.1.2	Double-line indented paragraphs .nP
OF	9.2.6	Odd-page footer .OF [arg]
OH	9.2.3	Odd-page header .OH [arg]
OK	6.6	Other keywords for Technical Memo cover sheet .OK [keyword] ...
OP	12.8	Odd page .OP
P	4.1	Paragraph .P [type]
PF	9.2.4	Page footer .PF [arg]
PH	9.2.1	Page header .PH [arg]

Table 16.A. Memorandum Macro Names
(continued)

MACRO	SECTION	DESCRIPTION
PM	9.9	Proprietary Marking .PM [code]
PX*	9.6	Page-header user exit .PX
R	12.1	Return to regular (Roman) font .R
RB	12.1	Roman/Bold .RB [Roman-arg] [bold-arg] [Roman] [bold] [Roman] [bold]
RD	12.12	Read insertion from terminal .RD [prompt] [diversion] [string]
RF	11.2	Reference end .RF
RI	12.1	Roman/Italic .RI [Roman-arg] [italic-arg] [Roman] [italic] [Roman] [italic]
RL	5.7	Reference list start .RL [text-indent] [1]
RP	11.4	Produce Reference Page .RP [arg] [arg]
RS	11.2	Reference start .RS [string-name]
S	12.9	Set **troff** formatter point size and vertical spacing .S [size] [spacing]
SA	12.2	Set adjustment (right-margin justification) default .SA [arg]
SG	6.11.1	Signature line .SG [arg] [1]
SK	12.7	Skip pages .SK [pages]
SM	12.10	Make a string smaller .SM string1 [string2] [string3]

Table 16.A. Memorandum Macro Names
(continued)

MACRO	SECTION	DESCRIPTION
SP	12.6	Space vertically .SP [lines]
TB	7.5	Table title .TB [title] [override] [flag]
TC	10.1	Table of contents .TC [slevel] [spacing] [tlevel] [tab] [head1] [head2] [head3] [head4] [head5]
TE	7.3	Table end .TE
TH	7.3	Table header .TH [N]
TL	6.2	Title of memorandum .TL [charging-case] [filing-case]
TM	6.4	Technical Memorandum number(s) .TM [number] ...
TP*	9.6	Top-of-page macro .TP
TS	7.3	Table start .TS [H]
TX*	10.1	Table of contents user exit .TX
TY*	10.1	Table of contents user exit (suppress CONTENTS) .TY
VL	5.8	Variable-item list start .VL text-indent [mark-indent] [1]
VM	9.8	Vertical margins .VM [top] [bottom]
WC	12.4	Footnote and Display Width Control .WC [format]

* Macros marked with an asterisk are not, in general, called (invoked) directly by the user. They are "user exits" defined by the user and called by the Memorandum Macros from inside header, footer, or other macros.

TABLE 16.B. String Names

STRING	SECTION	DESCRIPTION
BU	3.7	Bullet (**nroff**: +, **troff**: •)
Ci	10.1	Table of contents indent list Up to seven scaled arguments for heading levels
DT	6.8	Date (current date, unless overridden) Month, day, year (e.g., May 31, 1984)
EM	3.8	Em dash string Produces an em dash in the **troff** formatter and a double hyphen in **nroff**
F	8.1	Footnote number generator **nroff**: \u\\n+(:p\d **troff**: \v'-.4m'\s-3\\n+(:p\s0\v'.4m'
HF	4.2.2.4.1	Heading font list Up to seven codes for heading levels 1 through 7 3 3 2 2 2 2 2 (levels 1 and 2 bold, 3 through 7 underlined by **nroff** and italicized by **troff**)
HP	4.2.2.4.3	Heading point size list Up to seven codes for heading levels 1 through 7
Le	7.6	Title for LIST OF EQUATIONS
Lf	7.6	Title for LIST OF FIGURES
Lt	7.6	Title for LIST OF TABLES
Lx	7.6	Title for LIST OF EXHIBITS
RE	12.3	SCCS Release and Level of Memorandum Macros Release.Level (e.g., 15.129)
Rf	11.1	Reference number generator
Rp	11.4	Title for References
Tm	3.9	Trademark string Places "TM" ½ line above text that it follows
	12.10	Seven accent strings are also available.

Note 1: If the released-paper style is used, then (in addition to the above strings) certain BTL location codes are defined as strings and are needed only until the .MT macro is called. The following codes are recognized: **AK, AL, ALF, CB, CH, CP, DR, FJ, HL, HO, HOH, HP, IH, IN, INH, IW, MH, MV, PY, RD, RR, WB, WH,** and **WV.**

Note 2: Section 1.5 has notes on setting and referencing strings.

TABLE 16.C. Number Register Names

REGISTER	SECTION	DESCRIPTION
A *†	2.4	Handles preprinted forms and Bell System logo 0, [0:2]
Au	6.3	Inhibits printing of author information 1, [0:1]
C *†	2.4	Copy type (original, DRAFT, etc.) 0 (Original), [0:4]
Cl	4.4 10.1	Level of headings saved for table of contents 2, [0:7]
Cp	10.1	Placement of list of figures, etc. 1 (on separate pages), [0:1]
D *†	2.4	Debug flag 0, [0:1]
De	7.2	Display eject register for floating dislays 0, [0:1]
Df	7.2	Display format register for floating displays 5, [0:5]
Ds	7.1	Static display pre- and post-space 1, [0:1]
E *†	2.4	Controls font of the Subject/Date/From fields 1 (**nroff**) 0 (**troff**), [0:1]
Ec	7.5	Equation counter, used by .EC macro 0, [0:?], incremented by one for each .EC call.
Ej	4.2.2.1	Page-ejection flag for headings 0 (no eject), [0:7]
Eq	7.4	Equation label placement 0 (right-adjusted), [0:1]

* An asterisk attached to a register name indicates this register can be set only from the command line or before the macro definitions are read by the formatter.

† Section 1.5 has notes on setting and referencing registers. Any register having a single-character name can be set from the command line.

Table 16.C. Number Register Names
(continued)

REGISTER	SECTION	DESCRIPTION
Ex	7.5	Exhibit counter, used by .EX macro 0, [0:?], incremented by one for each .EX call.
Fg	7.5	Figure counter, used by .FG macro 0, [0:?], incremented by one for each .FG call.
Fs	8.4	Footnote space (i.e., spacing between footnotes) 1, [0:?]
H1-H7	4.2.2.5	Heading counters for levels 1 through 7 0, [0:?], incremented by the .H macro of corresponding level or the .HU macro if at level given by the **Hu** register. The H2 through H7 registers are reset to 0 by any .H (.HU) macro at a lower-numbered level.
Hb	4.2.2.2	Heading break level (after .H and .HU) 2, [0:7]
Hc	4.2.2.3	Heading centering level for .H and .HU 0 (no centered headings), [0:7]
Hi	4.2.2.2	Heading temporary indent (after .H and .HU) 1 (indent as paragraph), [0:2]
Hs	4.2.2.2	Heading space level (after .H and .HU) 2 (space only after .H 1 and .H 2), [0:7]
Ht	4.2.2.5	Heading type (for .H: single or concatenated numbers) 0 (concatenated numbers: 1.1.1, etc.), [0:1]
Hu	4.3	Heading level for unnumbered heading (.HU) 2 (.HU at the same level as.H 2), [0:7]
Hy	3.4	Hyphenation control for body of document 0 (automatic hyphenation off), [0:1]
L *†	2.4	Length of page 66, [20:?] (11i, [2i:?] in **troff** formatter)
Le	7.6	List of equations 0 (list not produced) [0:1]
Lf	7.6	List of figures 1 (list produced) [0:1]

Table 16.C. Number Register Names
(continued)

REGISTER	SECTION	DESCRIPTION
Li	5.3	List indent 6 (**nroff**) 5 (**troff**), [0:?]
Ls	5.1	List spacing between items by level 6 (spacing between all levels) [0:6]
Lt	7.6	List of tables 1 (list produced) [0:1]
Lx	7.6	List of exhibits 1 (list produced) [0:1]
N *†	2.4	Numbering style 0, [0:5]
Np	4.1	Numbering style for paragraphs 0 (unnumbered) [0:1]
O *†	2.4	Offset of page .75i, [0:?] (0.5i, [0i:?] in **troff** formatter) For **nroff** formatter, these values are unscaled numbers representing lines or character positions. For **troff** formatter, these values must be scaled.
Oc	10.1	Table of contents page numbering style 0 (lowercase Roman), [0:1]
Of	7.5	Figure caption style 0 (period separator), [0:1]
P†	2.4	Page number managed by Memorandum Macros 0, [0:?]
Pi	4.1	Paragraph indent 5 (**nroff**) 3 (**troff**), [0:?]
Ps	4.1	Paragraph spacing 1 (one blank space between paragraphs), [0:?]
Pt	4.1	Paragraph type 0 (paragraphs always left justified), [0:2]

Table 16.C. Number Register Names
(continued)

REGISTER	SECTION	DESCRIPTION
Pv	9.10	"PRIVATE" header 0 (not printed), [0:2]
Rf	11.1	Reference counter, used by .RS macro 0, [0:?], incremented by one for each .RS call.
S *†	2.4	The **troff** formatter default point size 10, [6:36]
Si	7.1	Standard indent for displays 5 (**nroff**) 3 (**troff**), [0:?]
T *†	2.4	Type of **nroff** output device 0, [0:2]
Tb	7.5	Table counter, used by .TB macro 0, [0:?], incremented by one for each .TB call.
U *†	2.4	Underlining style (**nroff**) for .H and .HU 0 (continuous underline when possible), [0:1]
W *†	2.4	Width of page (line and title length) 6i, [10:1365] (6i, [2i:7.54i] in the **troff** formatter)

* An asterisk attached to a register name indicates this register can be set only from the command line or before the macro definitions are read by the formatter.

† Section 1.5 has notes on setting and referencing registers. Any register having a single-character name can be set from the command line. [*head5*]

TABLE 16.D. Error Messages

Memorandum Macro Error Messages

An **mm** error message has a standard part followed by a variable part. The standard part has the form:

ERROR: (*filename*)input line *n*:

Variable part *n* consists of a descriptive message usually beginning with a macro name. They are listed below in alphabetical order by macro name, each with a more complete explanation.

ERROR MESSAGE	DESCRIPTION
Check TL, AU, AS, AE, MT sequence	The correct order of macros at the start of a memorandum is shown in section 6.1. Something has disturbed this order.
Check TL, AU, AS, AE, NS, NE, MT sequence	The correct order of macros at the start of a memorandum is shown in section 6.1. Something has disturbed this order. Occurs if the .AS 2 {6.5} macro was used.
CS:cover sheet too long	Text of the cover sheet is too long to fit on one page. The abstract should be reduced or the indent of the abstract should be decreased {6.5}.
DE:no DS or DF active	A .DE macro has been encountered, but there has not been a previous .DS or .DF macro to match it.
DF:illegal inside TL or AS	Displays are not allowed in the title or abstract.
DF:missing DE	A .DF macro occurs within a display, i.e., a .DE macro has been omitted or mistyped.
DF:missing FE	A display starts inside a footnote. The likely cause is the omission (or misspelling) of a .FE macro to end a previous footnote.
DF:too many displays	More than 26 floating displays are active at once, i.e., have been accumulated but not yet output.

Table 16.D. Error Messages
(continued)

ERROR MESSAGE	DESCRIPTION
DS:illegal inside TL or AS	Displays are not allowed in the title or abstract.
DS:missing DE	A .DS macro occurs within a display, i.e., a .DE has been omitted or mistyped.
DS:missing FE	A display starts inside a footnote. The likely cause is the omission (or misspelling) of a .FE to end a previous footnote.
FE:no FS active	A .FE macro has been encountered with no previous .FS to match it.
FS:missing DE	A footnote starts inside a display, i.e., a .DS or .DF occurs without a matching .DE.
FS:missing FE	A previous .FS macro was not matched by a closing .FE, i.e., an attempt is being made to begin a footnote inside another one.
H:bad arg: *value*	The first argument to the .H macro must be a single digit from one to seven, but *value* has been supplied instead.
H:missing arg	The .H macro needs at least one argument.
H:missing DE	A heading macro (.H or .HU) occurs inside a display.
H:missing FE	A heading macro (.H or .HU) occurs inside a footnote.
HU:missing arg	The .HU macro needs one argument.
LB:missing arg(s)	The .LB macro requires at least four arguments.
LB:too many nested lists	Another list was started when there were already six active lists.
LE:mismatched	The .LE macro has occurred without a previous .LB or other list-initialization macro {5.2.1}. This is not a fatal error. The message is issued because there exists some problem in the preceding text.

Table 16.D. Error Messages
(continued)

ERROR MESSAGE	DESCRIPTION
LI:no lists active	The .LI macro occurred without a preceding list-initialization macro. The latter probably has been omitted or entered incorrectly.
ML:missing arg	The .ML macro requires at least one argument.
ND:missing arg	The .ND macro requires one argument.
RF:no RS active	The .RF macro has been encountered with no previous .RS to match it.
RP:missing RF	A previous .RS macro was not matched by a closing .RF.
RS:missing RF	A previous .RS macro was not matched by a closing .RF.
S:bad arg: *value*	The incorrect argument *value* has been given for the .S macro {12.9}.
SA:bad arg: *value*	The argument to the .SA macro (if any) must be either 0 or 1. The incorrect argument is shown as *value*.
SG:missing DE	The .SG macro occurred inside a display.
SG:missing FE	The .SG macro occurred inside a footnote.
SG:no authors	The .SG macro occurred without any previous .AU macro(s).
VL:missing arg	The .VL macro requires at least one argument.
WC:unknown option	An incorrect argument has been given to the .WC macro {12.4}.

Table 16.D. Error Messages
(continued)

Formatter Error Messages

Most messages issued by the formatter are self-explanatory. Those error messages over which the user has some control are listed below. Any other error messages should be reported to the local system support group.

ERROR MESSAGE	DESCRIPTION
Cannot do ev	Can be caused by: • setting a page width that is negative or extremely short • setting a page length that is negative or extremely short • reprocessing a macro package (e.g., performing a **.so** request on a macro package that was already requested on the command line) • requesting the **troff** formatter **−s1** option on a document that is longer than ten pages.
Cannot execute *filename*	Given by the .! request if the *filename* is not found.
Cannot open *filename*	Indicates one of the files in the list of files to be processed cannot be opened.
Exception word list full	Indicates too many words have been specified in the hyphenation exception list (via **.hw** requests).
Line overflow	Indicates output line being generated was too long for the formatter line buffer capacity. The excess was discarded. Likely causes for this message are very long lines or words generated through the misuse of \c of the **.cu** request, or very long equations produced by **eqn/neqn**(1).
Nonexistent font type	Indicates a request has been made to mount an unknown font.

Table 16.D. Error Messages
(continued)

ERROR MESSAGE	DESCRIPTION
Nonexistent macro file	Indicates the requested macro package does not exist.
Nonexistent terminal type	Indicates the terminal options refer to an unknown terminal type.
Out of temp file space	Indicates additional temporary space for macro definitions, diversions, etc. cannot be allocated. This message often occurs because of unclosed diversions (missing .FE or .DE), unclosed macro definitions (e.g., missing ".."), or a huge table of contents.
Too many page numbers	Indicates the list of pages specified to the **−o** formatter option is too long.
Too many number registers	Indicates the pool of number register names is full. Unneeded registers can be deleted by using the **.rr** request.
Too many strings/macros	Indicates the pool of string and macro names is full. Unneeded strings and names macros can be deleted using the **.rm** request.
Word overflow	Indicates a word being generated exceeded the formatter word buffer capacity. Excess characters were discarded. Likely causes for this message are very long lines, words generated through the misuse of \c of the **.cu** request, or very long equations produced by **eqn/neqn**(1).