A/UX Text-Processing Tools

Release 3.0

LIMITED WARRANTY ON MEDIA AND REPLACEMENT

If you discover physical defects in the manuals distributed with an Apple product or in the media on which a software product is distributed, Apple will replace the media or manuals at no charge to you, provided you return the item to be replaced with proof of purchase to Apple or an authorized Apple dealer during the 90-day period after you purchased the software. In addition, Apple will replace damaged software media and manuals for as long as the software product is included in Apple's Media Exchange Program. While not an upgrade or update method, this program offers additional protection for up to two years or more from the date of your original purchase. See your authorized Apple dealer for program coverage and details. In some countries the replacement period may be different; check with your authorized Apple dealer.

ALL IMPLIED WARRANTIES ON THE MEDIA AND MANUALS, INCLUDING IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, ARE LIMITED IN DURATION TO NINETY (90) DAYS FROM THE DATE OF THE ORIGINAL RETAIL PURCHASE OF THIS PRODUCT.

Even though Apple has tested the software and reviewed the documentation, **APPLE MAKES NO WARRANTY OR REPRESENTATION, EITHER EXPRESS, OR IMPLIED, WITH RESPECT TO SOFTWARE, ITS QUALITY, PERFORMANCE, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. AS A RESULT, THIS SOFTWARE IS SOLD "AS IS," AND YOU, THE PURCHASER, ARE ASSUMING THE ENTIRE RISK AS TO ITS QUALITY AND PERFORMANCE.**

IN NO EVENT WILL APPLE BE LIABLE FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES RESULTING FROM ANY DEFECT IN THE SOFTWARE OR ITS DOCUMENTATION, even if advised of the possibility of such damages. In particular, Apple shall have no liability for any programs or data stored in or used with Apple products, including the costs of recovering such programs or data.

THE WARRANTY AND REMEDIES SET FORTH ABOVE ARE EXCLUSIVE AND IN LIEU OF ALL OTHERS, ORAL OR WRITTEN, EXPRESS, OR IMPLIED. No Apple dealer, agent, or employee is authorized to make any modification, extension, or addition to this warranty.

Some states do not allow the exclusion or limitation of implied warranties or liability for incidental or consequential damages, so the above limitation or exclusion may not apply to you. This warranty gives you specific legal rights, and you may also have other rights which vary from state to state.

Contents

1   Introduction to A/UX Text Processing / 1-1

5 ms Macros / 5-1

Figures and Tables

6 `me` Macros / 6-1

Chapter 5 ms Macros / 5-1

Chapter 6   me Macros / 6-1

Chapter 7   `tbl` Tables / 7-1

Chapter 8   `eqn` Equations / 8-1

Chapter 9   `pic` Line Drawings / 9-1

Chapter 10 `grap` Graphs / 10-1

About This Guide

Welcome to *A/UX Text-Processing Tools*. This book describes the commands you need to format text, tables, equations, and graphics. You can also use the tutorial in Chapter 2 if you need a quick brush-up on `troff` text processing. The companion book, *A/UX Text-Editing Tools,* presents detailed information on the five text editors provided by A/UX, and describes how to use the editors to create and edit text.

What are text-processing tools?

In A/UX you can use the UNIX® text-processing tools you're already familiar with: the formatters `troff` and `nroff`; the macro packages, `mm`, `me`, and `ms`; and a variety of preprocessors such as `pic`, `eqn`, `grap`, and `tbl`. Using these text-processing tools, you can design documents to suit your specific needs.

Who should use this book

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

How to use this book

This manual is meant to be used as a reference guide, but it also includes a tutorial. If your text-processing skills are rusty, you can work through the lessons in Chapter 2. You can use the table of contents to find the section that covers your general need and can use the index when you know exactly what command or process you want to refer to. For example, if you're formatting a paragraph using ms macros and you want to know what options are available, you could look in the table of contents for Chapter 5, "ms Macros," and find the section "Creating Paragraphs." However, if you know you want to create an indented paragraph, but you don't know what command to use, you would refer to "paragraphs, indenting" in the index.

A/UX Text-Processing Tools contains the following chapters:

- Chapter 1, "Introduction to A/UX Text Processing," gives a brief overview of the A/UX text-processing tools, explains page layout concepts, describes fonts, and introduces you to other formatting features.

- Chapter 2, "troff/mm Tutorial," guides you through three lessons: You'll produce a formatted letter, produce a letterhead, and modify the appearance of a page.

- Chapter 3, "nroff/troff Formatters," tells you how to use the powerful capabilities of nroff and troff formatters in A/UX.

- Chapter 4, "mm Macros," is a guide and reference for users of the memorandum macros.

- Chapter 5, "ms Macros," is a guide and reference for users of the ms macros, designed for writing general-purpose documents.

- Chapter 6, "me Macros," is a guide and reference for users of the me macros, designed for writing thesis papers at the University of California at Berkeley.

- Chapter 7, "tbl Tables," explains how tbl works and how you can use it to create tables that meet your specific needs.

- Chapter 8, "eqn Equations," explains how to use eqn to create typeset-quality mathematical text.

- Chapter 9, "pic Line Drawings," shows you how to create simple line drawings using the pic preprocessor.

- Chapter 10, "grap Graphics," is a guide to a graph-drawing program you can use to create charts and graphs.

- Chapter 11, "Related Tools," is a brief guide to additional text-processing tools.
- The glossary contains definitions of useful text-processing terms.

Conventions used in this guide

A/UX guides follow specific conventions. For example, words that require special emphasis appear in specific fonts or font styles. The following sections describe the conventions used in all A/UX guides.

Keys and key combinations

Certain keys on the keyboard have special names. These modifier and character keys, often used in combination with other keys, perform various functions. In this guide, the names of these keys are in Initial Capital letters followed by SMALL CAPITAL letters.

The key names are

CAPS LOCK	DOWN ARROW (↓)	OPTION	SPACE BAR
COMMAND (⌘)	ENTER	RETURN	TAB
CONTROL	ESCAPE	RIGHT ARROW (→)	UP ARROW (↑)
DELETE	LEFT ARROW (←)	SHIFT	

Sometimes you will see two or more names joined by hyphens. The hyphens indicate that you use two or more keys together to perform a specific function. For example,

Press COMMAND-K

means "Hold down the COMMAND key and then press the K key."

Terminology

In A/UX guides, a certain term can represent a specific set of actions. For example, the word *enter* indicates that you type a series of characters on the command line and press the RETURN key. The instruction

Enter ls

means "Type ls and press the RETURN key."

Here is a list of common terms and the corresponding actions you take.

Term	Action
Click	Press and then immediately release the mouse button.
Drag	Position the mouse pointer, press and hold down the mouse button while moving the mouse, and then release the mouse button.
Choose	Activate a command in a menu. To choose a command from a pull-down menu, position the pointer on the menu title and hold down the mouse button. While holding down the mouse button, drag down through the menu until the command you want is highlighted. Then release the mouse button.
Select	Highlight a selectable object by positioning the mouse pointer on the object and clicking.
Type	Type a series of characters *without* pressing the RETURN key.
Enter	Type the series of characters indicated and press the RETURN key.

The `Courier` font

Throughout A/UX guides, words that appear on the screen or that you must type exactly as shown are in the `Courier` font.

For example, suppose you see this instruction:

Type `date` on the command line and press RETURN.

The word `date` is in the `Courier` font to indicate that you must type it.

Suppose you then read this explanation:

After you press RETURN, information such as this appears on the screen:

```
Tues Oct 17 17:04:00 PDT 1989
```

In this case, `Courier` is used to represent the text that appears on the screen.

All A/UX manual page names are also shown in the `Courier` font. For example, the entry `ls`(1) indicates that `ls` is the name of a manual page in an A/UX reference manual. See "Manual Page Reference Notation" later in this preface for more information on the A/UX command reference manuals.

Font styles

Italics are used to indicate that a word or set of words is a placeholder for part of a command. For example,

cat *filename*

tells you that *filename* is a placeholder for the name of a file you want to display. For example, if you wanted to display the contents of a file named Elvis, you would type the word Elvis in place of *filename*. In other words, you would enter

cat Elvis

New terms appear in **boldface** where they are defined. Boldface is also used for steps in a series of instructions.

A/UX command syntax

A/UX commands follow a specific command syntax. A typical A/UX command gives the command name first, followed by options and arguments. For example, here is the syntax for the wc command:

wc [-l] [-w] [-c] [*filename*]...

In this example, wc is the command, -l, -w, and -c are options and *filename* is an argument. Brackets ([ ]) enclose elements that are not necessary for the command to execute. The ellipsis (...) indicates that you can specify more than one argument. Brackets and ellipses are *not* to be typed. Also, note that each command element is separated from the next element by a space.

The following table gives more information about the elements of an A/UX command.

Element	Description
command	The command name.
option	A character or group of characters that modifies the command. Most options have the form *–option*, where *option* is a letter representing an option. Most commands have one or more options.
argument	A modification or specification of a command, usually a filename or symbols representing one or more filenames.
[ ]	Brackets used to enclose an optional item—that is, an item that is not essential for execution of the command.
...	Ellipses are used to indicate that you can enter more than one argument.

For example, the `wc` command is used to count lines, words, and characters in a file. Thus, you can enter

```
wc -w Priscilla
```

In this command line, `-w` is the option that instructs the command to count all of the words in the file, and the argument `Priscilla` is the file to be searched.

Manual page reference notation

The *A/UX Command Reference*, the *A/UX Programmer's Reference*, the *A/UX System Administrator's Reference*, the *X11 Command Reference for A/UX*, and the *X11 Programmer's Reference for A/UX* contain descriptions of commands, subroutines, and other related information. Such descriptions are known as *manual pages* (often shortened to *man pages*). Manual pages are organized within these references by section numbers. The standard A/UX cross-reference notation is

command (section)

where *command* is the name of the command, file, or other facility; and *section* is the number of the section in which the item resides.

- Items followed by section numbers (1M) and (8) are described in the *A/UX System Administrator's Reference*.
- Items followed by section numbers (1) and (6) are described in the *A/UX Command Reference*.

- Items followed by section numbers (2), (3), (4), and (5) are described in the *A/UX Programmer's Reference.*

- Items followed by section number (1X) are described in the *X11 Command Reference for A/UX.*

- Items followed by section numbers (3X) and (3Xt) are described in the *X11 Programmer's Reference for A/UX.*

For example

```
cat(1)
```

refers to the command `cat`, which is described in Section 1 of the *A/UX Command Reference.*

You can display manual pages on the screen by using the `man` command. For example, you could enter the command

```
man cat
```

to display the manual page for the `cat` command, including its description, syntax, options, and other pertinent information. To exit a manual page, press the SPACE BAR until you see a command prompt, or type `q` at any time to return immediately to your command prompt.

For more information

To find out where you need to go for more information about how to use A/UX, see *Road Map to A/UX.* This guide contains descriptions of each A/UX guide and ordering information for all the guides in the A/UX documentation suite.

1 Introduction to A/UX Text Processing

The A/UX operating system provides a large number of tools for editing, formatting, and printing text and graphics. You can use these tools to prepare almost any kind or size of document, from newsletters to books. This chapter provides a conceptual overview of A/UX text processing. It describes what A/UX text-processing tools are, explains layout and font concepts, and gives a brief introduction to other formatting features that you might find useful. It also contains a short section on the printing process.

What are the A/UX text-processing tools?

To understand the A/UX text-processing tools, it is helpful to understand the process involved in producing a final printed document. The sequence typically looks something like that in Figure 1-1.
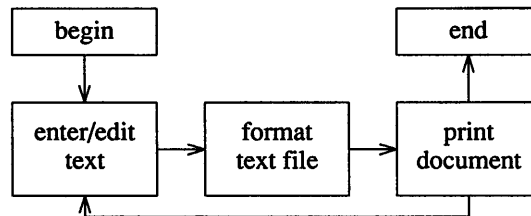
Figure 1-1 Producing a printed document

It is a basic assumption of the A/UX text-processing system that these tasks are separable from one another and ought to be handled by different programs. First, you use one of the standard A/UX editors to *enter* and *edit* your text. The editor doesn't format or print the file; it merely stores your text, exactly as you enter it. To arrange the text into pages and paragraphs, you use a *formatting* program (usually `troff` or `nroff` in conjunction with a macro package). These programs use instructions you have entered in the text file, which indicate how you want the final output to look. Once the text is edited and formatted, you may *print* the document by directing the formatted output to a printer.

`troff` and `nroff` for text

The A/UX text-processing system is based on a pair of programs called `troff` and `nroff`. `troff` formats its input for printing on any high-resolution typesetter or laser printer that is capable of printing multiple fonts and type sizes. `nroff` formats its input for printing on less-capable devices such as daisy wheel and dot matrix printers or your terminal screen. `troff` and `nroff` are for the most part compatible with each other, so that a single input file may be processed with either formatting program. `nroff` simply ignores any `troff` commands that the intended output device cannot support. From now on in this chapter, any reference to `troff` means either `nroff` or `troff`.

As mentioned above, troff searches through your file for commands. Input consists of text, which will print, and **commands,** which set parameters or call out special characters. These are troff commands. There are two ways to call out a command:

- By beginning a line with a **control character** (period or single quotation) optionally followed by a space or tab, followed by a one- or two-character command name, and then followed by a space or a new line. These are sometimes called *dot commands.* The single quotation suppresses the break function (the forced output of a partially filled line) caused by certain requests. Unrecognized command names are ignored.

- By typing an **escape character** (\), followed by a command name anywhere in a line. These are sometimes called "escape sequences."

The following are examples of troff dot commands:

```
.sp 4
.ft B
```

These instruct troff to leave four blank lines and switch into the bold font.

The following is an example of a troff escape sequence:

```
The last word on this line is \s20big.\s10
```

This command causes troff to produce the following output:

The last word on this line is **big**.

The sequence \s20 instructs troff to switch to point size 20. The same effect could be achieved using troff dot commands, as follows:

```
The last word on this line is
.ps 20
big.
.ps 10
```

The mm macro package for text

troff and nroff provide facilities for controlling virtually all features affecting the appearance of the final printed page. These programs do so, however, at a relatively low level; for instance, neither program provides automatic margins, page headers and footers, or page numbering. To obtain these features, as well as countless others you will probably need, you must use a macro package in conjunction with troff. A **macro** is a collection of troff commands grouped into a useful unit, and a **macro package** is a collection of macros grouped into a useful unit.

The standard A/UX macro package is called mm. (For a brief discussion of other macro packages, see "Other Macro Packages" later in this chapter.) The mm package provides two kinds of additions to basic troff capabilities:

■ a large number of dot commands that are not included in the troff command set but are necessary for most document processing

■ default parameter settings governing margins, page length, paragraph indent levels, and so forth

The mm dot commands are almost universally uppercase, to distinguish them from troff dot commands, which are all lowercase. For example, you can use

.P

to indicate the beginning of a paragraph. You use these additional dot commands exactly like troff dot commands. However, when you run the file through the formatting program, troff won't understand these macros unless you get it to read their definitions first. You can do this by invoking troff with the -mm argument:

troff -mm *file*

Thus, the argument to troff gives you access to the mm macro package. You can get access to other macro packages in the same way.

tbl for tables

It's easy to produce tables in a document by using the program tbl. Figure 1-2 shows an example of tbl output.

Text processing programs	
Program	**Function**
eqn	format equations
grap	format graphs
lp	printer spooler
nroff	low-quality output
pic	format pictures
tbl	format tables
troff	high-quality output
vi	enter/edit text

Figure 1-2 Example of tbl output

The **tbl** program, unlike the mm package, operates as a **preprocessor** to troff. tbl processes the input file containing table specifications before the file is processed by troff, as follows:

```
tbl file | troff -mm
```

This is because tbl translates the table specifications into troff commands. tbl recognizes these specifications when they occur between lines beginning with one of the commands .TS and .TE. For instance, the input for the table in Figure 1-2 looks like this in the text file:

```
.TS
box center tab(:) ;
c s
c c
lf7 l .
\f6Text Processing Programs\fR .
sp .5
eqn:format equations
grap:format graphs
lp:printer spooler
nroff:low-quality output
pic:format pictures
tbl:format tables
troff:high-quality output
vi:enter/edit text
.TE
```

For a complete discussion of the tbl program, see Chapter 7, "tbl Tables."

eqn for formatting equations

The A/UX text-processing system includes another troff preprocessor, eqn, that allows you to include mathematical equations and formulas in documents. eqn searches for equation specifications contained within .EQ and .EN pairs. For example, the input

```
.EQ
x + y = 4 sup 2
.EN
```

yields the output

$x+y=4^2$

And the input

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

yields the output

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

Like tbl, eqn is a preprocessor to troff. Its general command line looks like

```
eqn file | troff -mm
```

See Chapter 8, "eqn Equations," for further details.

pic for pictures

You may also produce simple line drawings in a document by using the **pic** program, another troff preprocessor. You specify pictures by including their descriptions within .PS and .PE pairs. For example, if you include the following description in the input file

```
.PS
box; arrow; ellipse
.PE
```

and run troff with the pic preprocessor

```
pic file | troff -mm ...
```

you get the picture shown in Figure 1-3.
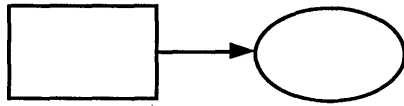
Figure 1-3 A simple picture

You can draw more complicated (and useful) drawings as well, such as those in Figures 1-4 and 1-5. The descriptions of these pictures are much more complicated than the simple description of Figure 1-3, but a mildly experienced `pic` user should have no trouble producing such diagrams. See Chapter 9, "`pic` Line Drawings," for a complete discussion of the `pic` language.
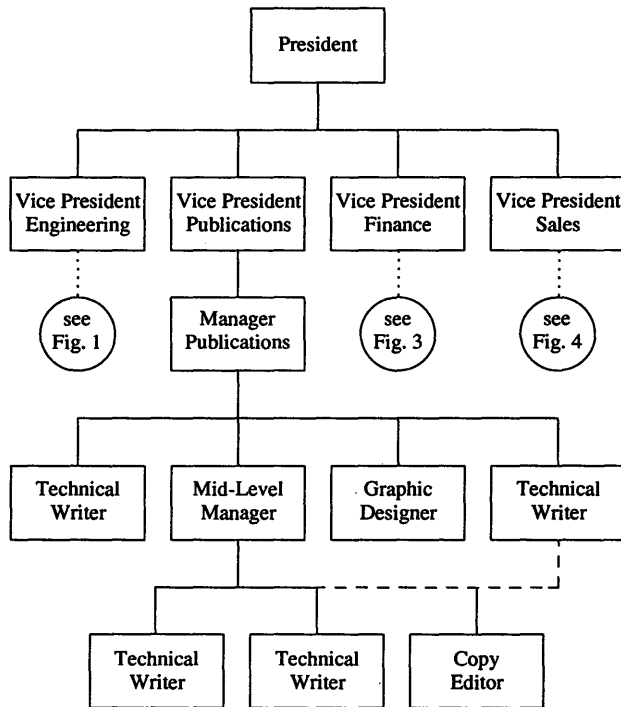
Figure 1-4 A more complicated picture

Ethernet
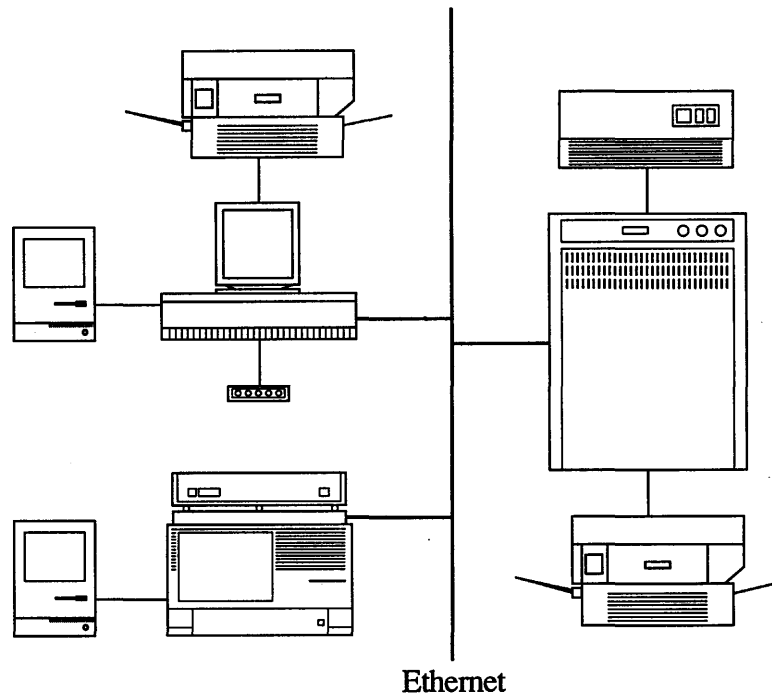
Figure 1-5 An even more complicated picture

grap for graphs

In addition to tables, equations, and simple line drawings, it is also possible to include graphs in a document formatted with troff. This is accomplished by using the **grap** preprocessor. Figure 1-6 is an example of grap output.

```
     sed
     vi
     grap
Text pic
processing tbl
programs eqn
     neqn
     troff
     nroff
     psdit
     lp
```

Program size (bytes)
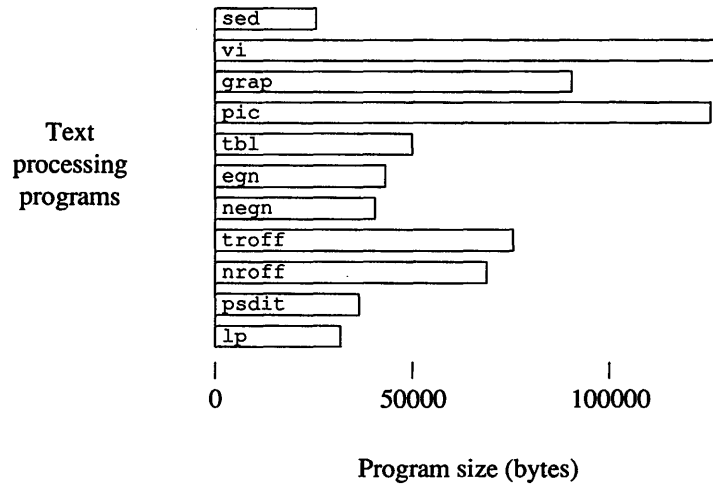
Figure 1-6 A graph

Like the other preprocessors, `grap` looks for a specification of how the graph should look and for the data to be graphed. These are enclosed within `.G1` and `.G2` pairs, as follows:

`.G1`

specification of graph

`.G2`

`grap`, however, is a preprocessor for `pic`; this means that `grap` translates the specification of the graph into `pic` code, not directly into `troff` code. So, to get graphical output, your command line must look something like this one:

```
grap file | pic | troff -mm
```

Figure 1-7 shows another example of `grap`'s capabilities. It charts San Francisco 49er wide receiver Jerry Rice's total receiving yardage per game for each of the sixteen regular season NFL football games in 1986. The height of the little football indicates the yardage, and the number inside the football indicates how many catches Rice made that day. Finally, the number of little goal posts under the football indicates how many touchdowns Rice scored in the game.
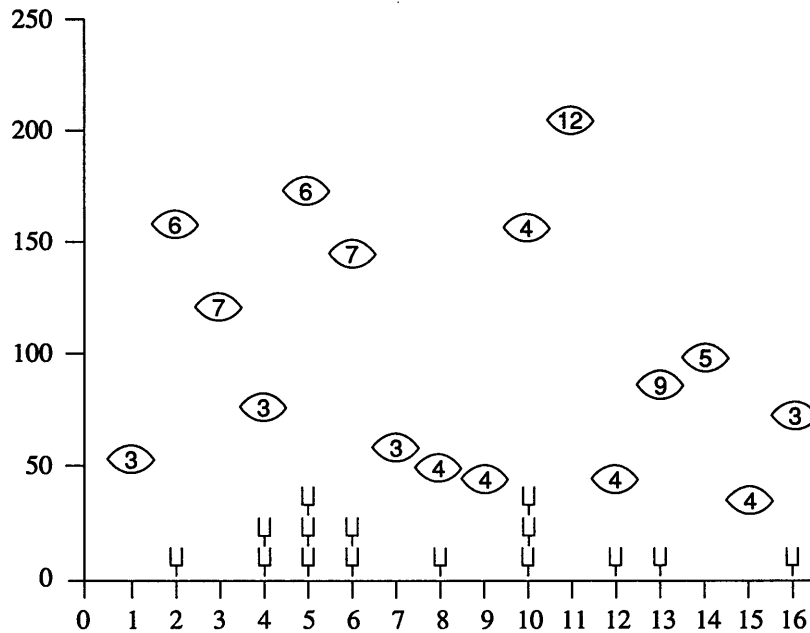
Jerry Rice's 1986 Season

Figure 1-7 A more complicated graph

For further information, see Chapter 10, "grap Graphs."

Other macro packages

In addition to the mm macro package, there are other macro packages that you may encounter on A/UX systems. Of particular note is the ms macro package (see Chapter 5, "ms Macros"). The ms program provides most of the same functions provided by the mm package, but with different syntax. For instance, a left-adjusted paragraph is indicated in ms with the macro

.LP

and in mm it is indicated with the macro

.P

For the most part, the page- and font-description concepts underlying the mm macros (described in the following two sections, "Page Layout Concepts" and "Font

Descriptions") will carry over into any other common macro package. Some mm macros, however, have no simple equivalent in other packages.

Another very common macro package is the man macro package. This collection of macros is intended for the special purpose of formatting manual pages as presented in *A/UX Command Reference, A/UX Programmer's Reference,* and *A/UX System Administrator's Reference.* See man(5) in *A/UX Programmer's Reference* for further details.

Page layout concepts

To get the most out of the A/UX text-processing programs, you must have some grasp of the terms used to describe page layout. This section introduces you to the most important of these.

If you use the mm macro package in conjunction with troff, the page is divided into a number of separate regions, some of which you can print on and some of which you cannot. The parts of a page are illustrated in Figure 1-8.

Generally, you cannot print on the entire physical page (typically a sheet of paper); the mm macros automatically generate margins on all four margins of the paper. You can, however, increase or reduce any of these margins independently of the others. In addition, the mm package automatically provides headers and footers (lines of text that are printed on the top and bottom, respectively, of every page). For more detailed discussion of these points, see "Margins" and "Headers and Footers" later in this chapter.

Entire physical page

↓

Left header   Center header   Right header

Title length

Top margin

Left margin   Right margin

Maximum line length

Bottom margin

Left footer   Center footer   Right footer

Printable portion of page
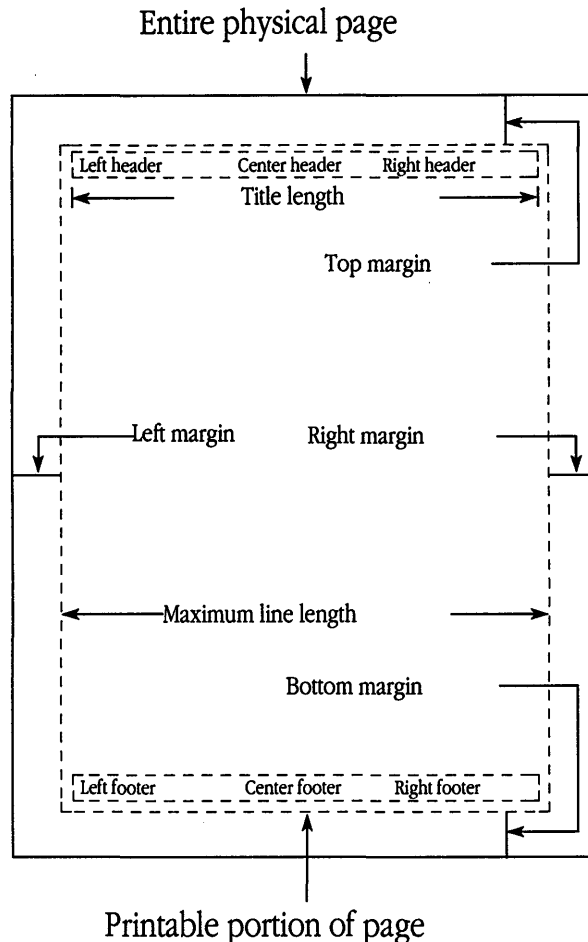
Figure 1-8 Parts of a page

Principal units of measurement

Many troff and mm commands require a unit of measure as part of the command. For instance, you must specify the line length as some number of inches or centimeters, and so on. troff and mm understand both inches and centimeters, as well as a number of other units that are more familiar to printers. (See Table 1-1.)

Table 1-1 Principal units of measurement

Unit	Abbreviation	Equivalence
Inch	i	None
Centimeter	c	2.54c = 1i
Pica	P	6P = 1i
Point	p	72p = 1i
Em	m	Width of "m" in current font
En	n	Width of "n" in current font

Of these units, only picas and points are likely to be unfamiliar to you. **Points** are used mostly to specify sizes of type (also called "point sizes"), and **picas** are often used for specifying line lengths and page lengths. For the most part, you can avoid using picas, but it is difficult to specify type sizes in any unit other than points.

Line length

The default line length using `troff` (with or without the mm macro package) is 6 inches. The maximum length of a line of text (or graphics) is the widest printable portion of the page, which is dependent on the capabilities of the printer you are using. You may specify the output line length with the `troff` command `.ll` followed by some measurement; for example,

```
.ll 7i
```

gives you a line length of 7 inches. There is no single mm command to accomplish the same thing. There is a number register that controls the length of the line and the page header and footer. You can set this register as follows:

```
.nr W 7i
```

For more information, see "Number Registers" later in this chapter.

Page length

The length of the physical page depends on the printer you are using; usually you will be working with one of the standard page sizes (for example, 8.5 by 11 inches, or A4). By default, the mm package assumes an 11-inch page, but you can alter the page length by setting the L number register:

```
.nr L 9i
```

The equivalent `troff` command is

```
.pl 9i
```

Note that this page length includes the top and bottom (vertical) margins. You can increase the amount of space taken by these margins with the `.vm` macro:

```
.VM 2 5
```

This adds two vertical spaces to the top margin and five vertical spaces to the bottom margin.

Paragraph types

You can specify more than one type of paragraph in a document. The mm macro package provides one macro, .P, for specifying the beginning of a paragraph (there is usually no need to specify the end of a paragraph). The argument you add to this macro determines the type of the paragraph. For instance, the command

```
.P 0
```

provides a left-adjusted paragraph, and the command

```
.P 1
```

provides a paragraph with the first line indented from the margin.

If there is no argument to the `.P` command, mm provides whatever you have selected as the default paragraph type. You select the default type with the command

```
.nr Pt n
```

where the argument n is as shown in Table 1-2.

Table 1-2 Argument *n* defaults

Argument	Resulting default
0	Left-adjusted
1	Indented
2	Indented except after headings, lists, or displays

Margins

There are two horizontal margins, one left and one right, on every page. The left margin is also known as the **page offset**, and you can change it using the `troff` command `.po`. The default is about 1 inch, but you can increase or decrease it.

The following command would be appropriate to center a 6-inch line of text on a piece of paper 8.5 inches wide:

```
.po 1.25i
```

You can change the right margin by changing the line length or the page offset.

Adjusted and filled text

By default, `troff` both fills and adjusts the text it formats. To **fill** text is to place as much text on a line as will fit, regardless of how the text occurs in the input file. One nice feature of `troff` is that it fills automatically. This means you can type your text into a file in whatever way is easiest for you to edit subsequently (for instance, beginning all sentences on a new line). `troff` may have to break a word in the middle to achieve a nice fit, but it will usually do this hyphenation in an intelligent manner.

You can control whether or not filling occurs with the `troff` commands `.nf` and `.fi`. For instance, the input

```
.nf
This text should not be filled.
So the output
will be arranged just like
the input.
```

produces the following output:

This text should not be filled.
So the output
will be arranged just like
the input.

You can turn filling back on with the `.fi` command.

To **adjust** text is to place small amounts of space between words in a filled line so that the line of output text is exactly the current line length. `troff` automatically adjusts text, but you can turn adjustment off with the `.na` command. You can turn adjustment back on with the `.na` command.

Indentation

Occasionally you need to indent some text to set it off from the surrounding text. You can do so with the `troff` command `.in`. For instance, the input

```
.P
This line is not indented at all.
.in .5i
This line is indented .5 inch.
.in 1i
This line is indented 1 inch.
.in 0
This line is not indented.
```

produces the following output:

This line is not indented at all.
                                    This line is indented .5 inch.
                                                            This line is indented 1 inch.
This line is not indented.

Notice that you can supply both absolute and relative arguments here, and that an argument of zero (0) returns to the current left margin. The indent persists until you reset it, or until it is reset automatically.

Headers and footers

A **header** is a line of text that is printed on the top of every page. Similarly, a **footer** is a line of text that is printed on the bottom of every page. (See Figure 1-8 for the locations of these lines.) Each of these lines is further divided into a left part, a center part, and a right part. You can specify any of these six items independently of the others. Further, you can specify different headers and footers for odd and even pages.

There are six mm macros affecting headers and footers:

.PH	page header (all pages)
.OH	odd header .i..OH macro
.EH	even header .i..EH macro (mm)
.PF	page footer (all pages) .i..PF macro (mm)
.OF	odd footer .i..OF macro (mm)
.EF	even footer .i..EF macro (mm)

Each of these macros takes the same kind of argument, a string surrounded by double quotation marks ("), with each of the three parts of the header or footer. For instance, we might specify a page header as follows:

```
.PH "'Chapter 8'%'The Bill of Rights'"
```

This header will appear on all pages. The left header will read "Chapter 8," the center header will be the page number, and the right header will read "The Bill of Rights."

Note that mm interprets the percent symbol specially in a header or footer specification; each time the header or footer is printed, the percent symbol is replaced by the current page number.

If you want one of the three parts of the header or footer to be empty, just leave the appropriate field in the argument string empty. For instance, the following command will cause the page number to be printed at the top of each page:

```
.PH "''%''"
```

If you need an apostrophe in the header or footer, you can change the delimiting character to anything you like, and mm will detect the change automatically. For instance, you might want the following header specification:

```
.PH "@Chapter 7@%@Bill's Alibi@"
```

You may specify a separate header or footer for odd and even pages. The following pair represents a very common way to handle headers:

```
.OH "@Chapter 7@%@Bill's Alibi@"
.EH "@Bill's Alibi@%@Chapter 7@"
```

Centered text

You can center a line of text on the page by using the troff dot command .ce. For example,

```
.ce
This line is centered.
```

produces

<div align="center">This line is centered.</div>

If you provide a numeric argument, the corresponding number of lines will be centered. For example,

```
.ce 3
This is the first centered line.
This is the second centered line.
This is the third and last centered line.
```

produces

<div style="text-align: center">

This is the first centered line.
This is the second centered line.
This is the third and last centered line.

</div>

Note that filling and adjusting are turned off for lines that are centered.

Footnotes

You can include footnotes in a document by enclosing the text to be included in the footnote between .FS and .FE pairs.[1] For example, the input

```
.FS
This is the text of a footnote.
It is smaller than the main text
and placed at the bottom of the page.
.FE
```

produces the footnote that appears at the bottom of this page. If you need consecutively numbered footnotes, you should include the string *F at the appropriate spot in the text. For further details about footnotes and footnote formats, see Chapter 4, "mm Macros."

Heading levels

In addition to the grouping provided by the paragraph macros, mm provides several macros for grouping paragraphs into sections and for generating a table of contents listing sections and subsections.

The primary macro for grouping paragraphs into sections is .H, for "heading level." A typical use of this macro might look like this:

```
.H 1 "The Clues to the Murder"
There was a broken window,
and the maid heard a loud scream
shortly before midnight.
In addition,
```

[1]This is the text of a footnote. It is smaller than the main text and placed at the bottom of the page.

The 1 indicates that a first-level heading is to be generated; mm automatically numbers these headings. If this is the fourth such macro in our text file, the output looks like this:

4. The Clues to the Murder

There was a broken window, and the maid heard a loud
scream shortly before midnight. In addition,

There may also be subsections within first-level sections. These are indicated with a second-level heading:

```
.H 2 "An Investigation of the Glass Shards"
```

The mm package allows for up to seven levels of headings (rarely are this many needed, however). In addition, there is a macro, .HU, for generating unnumbered headings:

```
.HU "Appendix A: Summary of Clues"
```

Many features of these heading-level macros, such as the point size and font for each heading level and the amount of spacing from surrounding text, can be adjusted to taste. See Chapter 4, "mm Macros," for a complete list of memorandum macros.

Font descriptions

troff is able to print in any font that is supported by the printer being used. nroff can generally print in only one font, but, depending upon the capabilities of the printer you are using, nroff may be able to simulate boldface by overstriking and italics by underlining.

Type families: Changing to bold and italic

You can achieve a great deal of clarity in a document by selecting fonts that are appropriate for your purposes. A **font** is a collection of letters and characters unified by a distinctive pattern or "look." What fonts are available to you is dependent on how troff has been configured, but typically at least the following three fonts are available:

Times Roman	ABCDEFGHIJKLMNOPQRSTUVWXYZ
Times Roman italic	*ABCDEFGHIJKLMNOPQRSTUVWXYZ*
Times Roman bold	**ABCDEFGHIJKLMNOPQRSTUVWXYZ**

By default, text is printed in "plain" Times Roman, unless you change fonts. You may change fonts with either a dot command (`.ft`) or an inline escape sequence (`\f`), followed by the name of the font desired. The following two lines give identical output:

```
This is in Times Roman,
.ft B
and this is Times Roman bold.


This is in Times Roman,
\fBand this is Times Roman bold.
```

The output in either case is

This is in Times Roman, **and this is Times Roman bold.**

You can also use mm macros (see Table 1-3).

Table 1-3 Using mm macros to change fonts to bold and italic

mm macro	Effect
`.B`	**Bold**
`.I`	*Italics*
`.R`	Roman

Thus, the example above could be further rewritten as

```
This is in Times Roman,
.B
and this is Times Roman bold.
```

You can also replace font names with numbers. For example, instead of \fB, you may write \f3. Many people prefer the numbers because it is easier to pick out the escape sequence. Which numbers correspond to which fonts depends on how your printer and software have been configured. For example, systems using the TranScript troff-to-PostScript® translator driving the Apple LaserWriter printer have the correspondence shown in Table 1-4.

Table 1-4 Using numbers to specify fonts

Number	Font
1	Times Roman
2	*Times Italic*
3	**Times Bold**
4	***Times Bold Italic***
5	Helvetica
6	**Helvetica Bold**
7	`Courier`
8	**`Courier Bold`**

Point size

troff can work with virtually any text size that the printer supports. The program is usually configured to allow you access to only a portion of those actually printable. Point sizes normally range approximately from 2 point to 80 point. (Point size 2 is so small that it's unreadable.) The following shows point size 80:

80

The default type size is 10 point. You may change point sizes in a variety of ways. Usually this is done with the `.ps` command:

```
.ps 14
This text is now in 14 point.
```

This produces

This text is now in 14 point.

You may also use the inline escape sequence \s. The input

```
This is in 10 point, \s14and this is in 14\s0.
```

produces

This is in 10 point, and this is in 14.

Notice that \s0 returns to the previous type size, not size 0.

Type size changes may also be specified relatively. For instance, you may rewrite the previous example as follows:

```
This is in 10 point, \s+4and this is in 14\s0.
```

Vertical spacing

The **vertical spacing** between two lines of text is the distance from the base of the characters on one line of text to the base of the characters on the next line. Normally, the vertical spacing is set to 12 points, which is enough to accommodate a 10-point character plus a small amount of white space between lines. If you change point sizes, you must increase or decrease the vertical spacing accordingly. You can change the vertical spacing with the `.vs` command:

```
.ps 20
.vs 22
```

A common mistake is to increase the point size without increasing the vertical spacing. In such a case you usually end up with garbage, for example,

This is 24-point text
at the normal 12-point
vertical spacing.

You can set both the point size and the vertical spacing at once with the mm macro
.S. For instance,

```
.S 24 26
```

sets the point size to 24 points and the vertical spacing to 26 points.

Character set

The set of characters that you can print using troff depends on the abilities of the printer you are using. Generally, a character is accessible to troff if it is a member of some font that troff knows about. A troff font typically includes the following characters:

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
a b c d e f g h i j k l m n o p q r s t u v w x y z
1 2 3 4 5 6 7 8 9 0
& . , : ; ' ' ~ ! @ # $ % ^ * ( ) - + = { } [ ] \ | > <

In addition, there may be other fonts known as "special" fonts. Originally these fonts were used for mathematical symbols not available on the standard Times Roman font, but a special font can contain any sort of characters or glyphs. A typical mathematical special font provides the following characters, which include a full Greek alphabet:

Α Β Ξ Δ Ε Φ Γ Θ Ι Κ Λ Μ Ν Ο Π Ψ Ρ Σ Τ Υ Ω Χ Η Ζ
α β ξ δ ε φ γ θ ι κ λ μ ν ο π ψ ρ σ τ υ ω χ η ζ
≠ ∉ ⊆ ´∇ − ∞ ⇒ ⇔ ⊇ ∫ ⟨ ∧ ⇐ ∨ ∈ ´ × ¬ ∂
+ − ∝ ⟩ ⇒ ° _ ⊂ " ⊃ □ √ ∃ ∀ ~ ≈

There are two standard ways to get one of these characters to print in a document. First, you can use a feature of the preprocessor eqn that allows inline equations. In that case, you would use the eqn name of these symbols. For instance, we have seen that eqn translates the word into the symbol ∫ (appropriately scaled, of course).

A second way to get access to special font characters is to use their troff name (see Table 1-6).

For a complete list, see Chapter 3, "nroff/troff Formatters."

Table 1-5 Accessing a few special font characters

Input	Output	Name
\(pl	+	Plus
\(mi	−	Minus
\(mu	¥	Multiplication
\(sr	÷	Square root ÷ (square root sign) 1-
\(br	|	Box rule
\(ua	⇌	Up arrow
\(da	Ø	Down arrow
\(ci	O	Circle
\(!=	Π	Not equal
\(is	∫	Integral

Accents

The mm macro package provides the ability to print accent marks over certain characters. To do this, you need to put the mm name of the accent mark after the letter you want accented. For example, the input

```
re\*'sume\*'
```

produces the word "résumé." The following accents are available:

Input	*Output*	*Name*
*`	`	grave accent
*´	´	acute accent
*^	^	circumflex
*~	~	tilde
*,	,	cedilla
*:	¨	umlaut (lowercase)
*;	¨	umlaut (uppercase)

Overstriking

The troff formatter provides one additional way of generating characters that are not in its basic character set: by overstriking two or more characters. The inline escape sequence \o will overstrike whatever characters (up to nine) are enclosed within single quotation marks.

The \o sequence centers each character as it overstrikes it. If instead you want the characters lined up on their left sides, you could use the \z escape sequence. This instructs troff to print the character that follows but not to move to the right after printing it.

Other formatting features

troff and the mm macro package provide several additional features that are very useful in document production: displays, automatic list and table of contents generation, multicolumn output, strings, and number registers.

Displays

Occasionally a certain stretch of text should be kept together on one page. For instance, it is generally preferred that the information in a table not be split across page breaks. tbl does not provide the service of preventing bad text breaks, but mm provides a way of doing it with displays. A **display** is a block of text that is to be kept on one page.

You can indicate a display by enclosing the relevant text within the pair of macros .DS and .DE, as follows:

```
.DS
This text will be kept all together.
No heading macros are allowed in a display,
but paragraph macros and lists are allowed.
By default the text of a display is not
filled or adjusted, but you can override
this by providing an argument
to the .DS macro.
.DE
```

If there is not enough space remaining on the page to fit this entire block, `troff` will begin a new page so that the block remains together.

Lists

Occasionally you want to provide a list of items. The mm package provides a number of macros designed to facilitate printing lists of various kinds. For instance,

```
.P
The remaining suspects are
.sp .5
.BL
.LI
Tim
.LI
Joe
.LI
the butler
.LI
the maid
.LE
.sp
```

produces

```
The remaining suspects are
```

- Tim
- Joe
- the butler
- the maid

The macro `.BL` is a **list-initialization macro**; it instructs mm that a bulleted list follows. The macro `.LI` indicates the beginning of each list item, and the macro `.LE` indicates the end of the list.

There are a number of other list-initialization macros:

.AL	Numbered or lettered list
.BL	Bulleted list
.DL	Dashed list
.ML	Marked list
.RL	Reference list
.VL	Variable-item list

As you would expect, the format of the list can be adjusted as needed; see Chapter 4, "mm Macros," for details.

Tables of contents

mm is able to generate a table of contents for your document by remembering all section headings and the pages where they occur as it formats the document. To get the table of contents printed, you must include the following macro at the end of your input file:

`.TC`

This macro causes mm to print out the accumulated section headings and page numbers. You may control the appearance of the table by adding arguments to the macro (see Chapter 4, "mm Macros").

Multicolumn output

By default, `troff` outputs the text in one column. You can instruct it to print two columns with the `.2c` macro.

To return to one column, use the `.1c` macro.

Strings

A **string** is a sequence of characters grouped together under a name. The mm macro package provides several predefined strings that you can use. For instance, the string * (DT will be replaced by the current date, as follows:

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

This results in

```
Today is September 7, 1990.
```

You get access to a string by preceding its name with the sequence * ((or, as we saw above, with the sequence * if the name of the string is only one character). In addition, you may define your own strings with the troff command .ds. Defining your own strings might be useful for abbreviating an often-used but lengthy phrase. For example,

```
.ds CU Pig Farmers of America Credit Union
.P
The annual board meeting of the \*(CU
was called to order at 2:11 p.m.
Chairman Curley reported
an unexpected rise
```

produces

The annual board meeting of the Pig Farmers of America Credit Union was called to order at 2:11 p.m. Chairman Curley reported an unexpected rise

Number registers

troff keeps track of many of the parameters governing the page layout by storing them in **number registers.** You may think of a number register as a slot having both a label (the name of the register) and something inside it (the value of the number register). Some of these registers are created and manipulated by troff and mm themselves, but you may also define your own number registers.

You can create a number register with the command .nr:

```
.nr YR 86
The profit in year 19\n(YR was $250,000.
```

In the text, you must precede the number register (here, YR) with \n. The value you define in the number register then appears in the output:

The profit in year 1986 was $250,000.

A more typical use of the .nr command is to change built-in parameters. For instance, you can use the command

```
.nr Pi 10
```

to change the paragraph indent to 10 ens. See Chapter 4, "mm Macros," for a complete list of number registers.

Defining and using macros

If you find yourself repeating the same sequence of troff commands, or almost the same sequence, you may find it useful to define a macro encapsulating that sequence of commands. You define a macro with the .de macro, for instance,

```
.de QP
.in +5n
.ll -10n
.ps -2
..
```

The line consisting of two dots indicates the end of the macro. Here we have defined a rudimentary quote paragraph macro: it indents the text from both sides and reduces the point size by 2.

You can also define macros with arguments, like many of the mm macros. The arguments are indicated in the definition with the sequences \\$1, \\$2, and so on. For example,

```
.de XX
Today is \\$1 the \\$2.
..
.XX Friday 6th
```
yields

Today is Friday the 6th.

Macro names should be chosen carefully to avoid conflicts with predefined mm macro names. To be safe, user-defined macros should be two characters with the first lowercase and the second uppercase. For example,

```
.de mN
```

Horizontal and vertical line spacing

troff includes commands for making arbitrary motions in a horizontal (\h) or vertical (\v) direction. For example,

```
There is a gap \h'0.5i' in this sentence.
```
yields

There is a gap   in this sentence.

Both \h and \v require a distance specification within single quotation marks; the two escape sequences \u and \d, however, move up and down a fixed distance and so require no argument. For example,

```
This sentence contains a superscript\u1\d.
```
yields

This sentence contains a superscript[1].

The TranScript package

As indicated earlier in this chapter, a printer interface program is needed to translate the output of `troff` into a form that is understood by your printer. If you wish to produce output on an Apple LaserWriter, you must pipe the output of `troff` through a program that translates it into PostScript, the page-description language used by the LaserWriter. For this purpose, the A/UX system contains a package of programs called TranScript.

The most important program in this package is `psdit`, which translates `troff` output into PostScript. For instance, the command line used in producing this chapter was

```
grap chap.1 | pic | tbl | eqn | troff -Tpsc -mm | psdit | lp
```

The only thing new here, aside from the postprocessor `psdit`, is the `-Tpsc` option to `troff`. This tells `troff` which type of printer it should format its output for; `troff` needs this information so that it can know which point sizes are legal for that printer and which fonts are available on the printer (among other things). The *psc* stands for "PostScript device" and is the appropriate option for the LaserWriter.

For more information on the TranScript package, `consult transcript(1M)` in *A/UX System Administrator's Reference.*

Line drawing

There are two `troff` commands for drawing horizontal and vertical lines, `\l` and `\L`. For example,

`\l'0.5i'\L'0.5i'`

prints

```
 ┌
─┘ │
   │
```

Document printing

`troff` produces output that is device independent. This means that you will need to process the output of `troff` with a program (usually called an *interface program*) that translates this output into a form that the printer understands. This step of the printing process may be done automatically, or you may need to invoke this program yourself. Check with local administrators to see what is appropriate for your installation. On the A/UX system, an interface program is provided to allow `troff` output to be printed on the LaserWriter; this program is called `psdit` and is discussed later in this chapter in "The TranScript Package."

Output devices

The A/UX family of text-processing tools is designed to be as independent of any particular type of **output device** as possible, thereby allowing the user to get output on any of a wide number of printers or display devices. On the high end of the spectrum, `troff` is capable of producing output on modern digital typesetters and phototypesetters, and on laser-driven printers, whose quality approaches that of much more expensive typesetters. `troff` can also send output to certain high-resolution video display terminals. On the low end of the spectrum, `nroff` can format its input for output on virtually any terminal screen, dot-matrix printer, or daisy-wheel printer.

When you print the letter, the name and address print out as follows:

Ms. Pandora S. Bach
Comparative Surveys, Inc.
79 Downing Street
San Jose, California 95128

Creating spaces between paragraphs

You can leave a space and a half on the printed page between the address and the salutation by using `.P`, the paragraph macro. Type

```
.P
```

on the line below `.DE`, and follow it with

```
Dear Ms. Bach:
```

on the next line, followed with another `.P` on the line after that. The file now looks like

```
.DS
Ms. Pandora S. Bach
Comparative Surveys, Inc.
79 Downing Street
San Jose, California 95128
.DE
.P
Dear Ms. Bach:
.P
```

where `.P` stands for "paragraph." Use the paragraph macro wherever you want to leave extra space or start a paragraph.

Creating a list with bullets

The body of this letter lists three items. To print them out in a bulleted list, with each item preceded by a bullet and indented five spaces, use the bulleted list macro. Starting at the line below the second .P, type

```
.P
Enclosed please find the following items:
.BL 5
.LI
A copy of a message from Ms. Gail Smith
dated March 6.
.LI
A copy of the worksheet you requested.
.LI
A \f(BIComparative Surveys\fR records
form and relevant information.
.LE
.P
Thank you for your attention to this account.
.P
```

Printing the file produces the following output:

Enclosed please find the following items:

- A copy of a message from Ms. Gail Smith dated March 6.

- A copy of the worksheet you requested.

- A *Comparative Surveys* records form and relevant information.

Thank you for your attention to this account.

Changing fonts

Note that in the text above, the phrase "Comparative Surveys" prints out in ***bold italic*** and the words after in roman. This is caused by the `troff` commands `\f(BI` and `\fR`.

The first command

```
\f(BI
```

instructs the printer to print the following text in ***bold italic Times Roman*** font.

The second command

```
\fR
```

instructs the printer to print the following text in **Times Roman** font.

Indenting text

To finish off your letter, you can use the indent command (`.in`) to print text indented on the page. Type

```
.in +2i
Sincerely yours,
.sp 3
John C. Doe
.in -2i
.sp
Enclosures
```

Printing the file produces the following output:

              Sincerely yours,

              John C. Doe

Enclosures

Formatting and printing your file

When you have entered all the above text and commands in your file `letter`, save the file on disk and exit `vi`. When you see the shell prompt on your screen again, you are ready to format your file and send it to the printer. (See *Setting Up Accounts and Peripherals for A/UX* for information about setting up a printer.)

At the shell prompt, type

```
troff -Tpsc -mm letter | psdit | lp
```

This command line sends your file through the `troff` program and mm macros, then sends it to a postprocessor, `psdit`, that prepares it for the LaserWriter, and finally sends it to the printer. See Chapter 1, "Introduction to A/UX Text Processing," and the reference chapters that follow for more information.

When the printer has received your file, you will see a message on your screen. Figures 2-1 and 2-2 show your file `letter` as it appears on your screen and on the printed page that is produced.

```
.DS
Ms. Pandora S. Bach
Comparative Surveys, Inc.
79 Downing Street
San Jose, California 95128
.DE
.P
Dear Ms. Bach:
.P
.P
Enclosed please find the following items:
.BL 5
.LI
A copy of a message from Ms. Gail Smith dated March 6.
.LI
A copy of the worksheet you requested.
.LI
A \f(BIComparative Surveys\fR
records form and relevant information.
.LE
.P
Thank you for your attention to this account.
.P
.in +2i
Sincerely yours,
.sp 3
John C. Doe
.in -2i
.sp
Enclosures
```

Figure 2-1 Contents of your file with text and troff/mm code

Ms. Pandora S. Bach
Comparative Surveys, Inc.
79 Downing Street
San Jose, California 95128

Dear Ms. Bach:

Enclosed please find the following items:

* A copy of a message from Ms. Gail Smith dated March 6.

* A copy of the worksheet you requested.

* A *Comparative Surveys* records form and relevant information.

Thank you for your attention to this account.

Sincerely yours,

John C. Doe

Enclosures

Figure 2-2 File printed on a LaserWriter

Lesson 2: Producing letterhead

To create letterhead stationery, you may first create a new file by invoking one of the A/UX text editors such as `vi`. Create the new file, `letterhead`, by entering

```
vi letterhead
```

Once you have opened the new file, you can use `vi` commands to enter text and `troff` and mm commands to format it.

This simple letterhead will consist of John Doe's name and address at the top of a page. Because of the physical size of this manual, the stationery will print out smaller than standard 8.5-by-11-inch paper. In "Changing the Size of Your Page" later in this chapter you will see how to change the code to print out a larger version of this letterhead.

Setting top-of-page instructions

The `troff` program uses several internal defaults to define how text will print out. You can change these defaults to fine-tune the format of your printed page.

For example, `troff` prints a page number at the top of each page. To prevent this, you can change the "page header" macro's definition. The page header macro accepts three fields: the left side of the page, the center, and the right side. In the definition, the three fields are separated by single quotation marks.

At the top of the file, enter

```
.PH "'''''"
```

This defines all three fields as empty.

You may define how many spaces are left at the top of the page, using the definition

```
.de TP
.sp 2
..
```

This tells the printer to start printing text two spaces below the default of 1 inch. Enter this definition in the file below the page header macro.

Changing the size of your text

The `troff` program uses point size 10 by default. This is the point size used in this manual. If you want the text of your letter (and any text in your letterhead) to appear in point size 10, you don't need to specify this to `troff`. However, if you want the text to appear slightly larger, for example, point size 11, you can use the mm command

```
.S 11 13
```

This changes the default point size to 11 and the vertical spacing to 13.

Changing the size of your page

Because of the physical size of this manual, the stationery in this tutorial will print out smaller than standard 8.5-by-11-inch paper. The length of a line of text, the width of the margin, and the length of the page itself are defined using number registers. Number registers are assigned values as follows:

```
.nr W 4i
```
specifies a 4-inch line

```
.nr O 2i
```
specifies 2-inch margins

```
.nr L 11i
```
specifies an 11-inch page

The W number register stands for the width of the text, and the O register stands of the offset from the physical width of the page.

To print out a standard-size page, change these definitions as follows:

```
.nr W 6i
```
Specifies a 6-inch line

```
.nr O 1i
```
Specifies 1-inch margins

```
.nr L 11i
```
Specifies an 11-inch page

Designing your letterhead

Enter the following commands in your file:

```
.sp
\l'4i'
.sp
\s14John C. Doe\s0
.br
 \l'4i'
.sp -1.75m
\l'4i'
.sp .25
.tl '''\s9\&P.O. Box 14, Carter, CA 94530\s0'
.sp
.tl '''\*(DT'
.sp 2
```

These commands are listed below with comment lines that describe what each one tells the printer to do.

`.sp`	Leave one blank line.
`\l'4i'`	Draw a line 4 inches long.
`.sp`	Leave one blank line.
`\s14John C. Doe\s0`	Print this text in point size 14.
`.br`	Break line here (go to next line).
`\l'4i'`	Draw a line 4 inches long.
`.sp -1.75m`	Go back up 1.75 em units.
`\l'4i'`	Draw a line 4 inches long.
`.sp .25`	Leave 1/4 vertical space.
`.tl '''\s9\&P.O. Box 14, Carter, CA 94530\s0'`	Print this text in point size 9, on the right side of the line.
`.sp`	Leave one blank line.
`.tl '''\*(DT'`	Print the current date on the right side of the line.
`.sp 2`	Leave two blank lines.

Note that the string `\*(DT` will print the current date (the date on which you format your letter). The "title" request:

`.tl ''''`

is similar to the page header macro described above in that it defines three separate fields, enclosed in single quotation marks. The three fields are the left side of the page, the center, and the right side. In the letterhead definition above, the title request is used to justify a string of text on the right side of the page.

If you format your `letterhead` file using the `troff` command line shown under "Formatting and Printing Your File" earlier in this chapter, your letterhead looks like the output in Figure 2-3.

John C. Doe

P.O. Box 14, Carter, CA 94530

August 28, 1987

Figure 2-3   A sample letterhead

Printing your letter on letterhead

Now that you have created a file containing the `troff` and mm codes to produce
letterhead stationery, save the file on disk and exit `vi`. You can now use this letterhead
with any letters you write by formatting it on the same command line as your letter.
Because the letterhead must print *before* the text of your letter, the command line should
look like this:

```
troff -Tpsc -mm letterhead letter | psdit | lp
```

This command line sends both files through the `troff` program and mm macros. The
letter this produces looks like the one in Figure 2-4.

John C. Doe

P.O. Box 14, Carter, CA 94530

August 28, 1987

Ms. Pandora S. Bach
Comparative Surveys, Inc.
79 Downing Street
San Jose, California 95128

Dear Ms. Bach:

Enclosed please find the following items:

* A copy of a message from Ms. Gail Smith dated March 6.

* A copy of the worksheet you requested.

* A *Comparative Surveys* records form and relevant information.

Thank you for your attention to this account.

Sincerely yours,

John C. Doe

Enclosures

Figure 2-4 A sample letter

Lesson 3: Modifying the appearance of a page

Now that you have created a simple letter and printed it out on letterhead stationery, you may want to modify the letter to include more information. In this lesson, you will learn how to produce a footnote and line graphics in your letter.

Producing a footnote

To include a footnote in your file named `letter`, first open the file using an editor such as `vi`:

```
vi letter
```

Move your cursor to the place in the file where you want the footnote to be referenced. This example uses a "dagger" symbol rather than a number. For example, move to the line in your file that reads

```
A copy of the worksheet you requested.
```

and place the dagger symbol at the end of the line:

```
A copy of the worksheet you requested.\(dg
```

When you include a footnote in your text, use the mm footnote macros. `.FS` stands for "footnote start" and `.FE` for "footnote end." These should be placed as close as possible to the footnote reference (in this case, `\(dg`). On the next line in your file, type

```
.FS \(dg
Note that the worksheet is dated March 20.
.FE
```

Your letter will look like the one in Figure 2-5.

John C. Doe

P.O. Box 14, Carter, CA 94530

August 28, 1987

Ms. Pandora S. Bach
Comparative Surveys, Inc.
79 Downing Street
San Jose, California 95128

Dear Ms. Bach:

Enclosed please find the following items:

• A copy of a message from Ms. Gail Smith dated March 6.

• A copy of the worksheet you requested.†

• A *Comparative Surveys* records form and relevant information.

Thank you for your attention to this account.

Sincerely yours,

John C. Doe

Enclosures

† Please note that the worksheet is dated March 20.

Figure 2-5 A sample letter with a footnote

Producing graphics

You can include simple line drawings in a document by using the preprocessor `pic` after you've entered appropriate picture specifications in your file.

Graphics can be useful in documents. For example, you might want to order some printed envelopes to go along with your custom stationery. A good way to let the printer know how you want it to look is to enclose a picture of the printed envelope. You can specify such a picture by including the following input in your file:

```
.PS
A: box ht 2i wid 4i
line from A.nw to A.c
line from A.ne to A.c
box invis ht .75i "John C. Doe" "P.O. Box 14"\
"Carter, CA" "94350" with .n at A.n
.PE
```

You can then process this with the command line

```
pic letter | troff -Tpsc -mm | psdit | lp
```

The output, in part, will look like Figure 2-6.
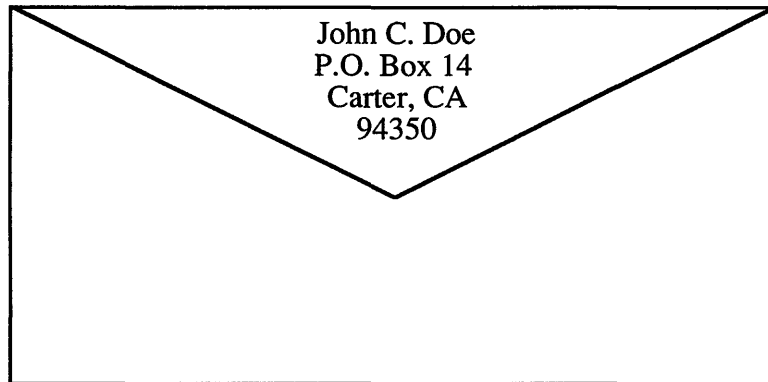
Figure 2-6 A sample line graphic

3 `nroff/troff` Formatters

This chapter introduces you to the capabilities of the `nroff/troff` formatters.

What is `nroff`/`troff` formatting?

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

The `troff` formatter formats text destined to be printed on a phototypesetter but intended to be converted by a postprocessor into codes that will drive a particular phototypesetter.

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

Because `nroff` and `troff` formatters are reasonably compatible, it is usually possible to prepare input acceptable to both. Conditional input is provided that enables you to embed input expressly destined for either program (see "Conditional Acceptance of Input"), 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 spacing. `nroff` does not have fractional space capabilities. For example, `nroff` will ignore the `troff` vertical space request `.sp .5` and will treat `.sp 1.3` as one space. 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` (see "Moving Characters Within a Line: Setting Local Motion" later in this chapter).

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` text formatter is a program that can drive virtually any phototypesetter because its output is an ASCII code describing the position, font, size, and so on of characters to be typeset on a page. This output must be converted by another program, called a postprocessor, into codes a particular phototypesetter will understand. Parameters such as fonts, character sizes, and special characters depend on the phototypesetter being driven.

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, and so on) are provided by the `troff` processor. It also provides macros, arithmetic variables and operations, and conditional testing for complicated formatting tasks.

Options when invoking `nroff` and `troff` formatters

The general form of invoking an `nroff` or `troff` formatter at the A/UX operating-system command level is

`nroff` [*options*] [*files*]

or

`troff` [*options*] [*files*]

where *options* represents any of a number of flag options 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 filename corresponding to the standard input. Input is taken from the standard input if no filenames are given. Options may appear in any order but must appear before the files. (See Table 3-1.)

Table 3-1 Options for invoking `nroff/troff`

Option	Effect
`-a`	(`troff` only.) 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.
`-e`	(`nroff` only.) Produce equally spaced words in adjusted lines using full terminal resolution.
`-F`*dir*	Get access to font information from the directory `dir/dev`*name,* where *name* is the default output device. The default font information directory is `/usr/lib/font/dev`*name.*
`-h`	(`nroff` only.) 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.
`-i`	Read standard input after the input files are exhausted.
`-m`*name*	Prefix the `/usr/lib/tmac/tmac.`*name* macro file to the input files. Multiple −m macro package requests on a command line are accepted and are processed in sequence.
`-n`*n*	Number the first generated page *n.*
`-o`*list*	Print only pages whose page numbers appear in *list,* which can consist of comma-separated numbers, number ranges, or both: ■ 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.
`-q`	Invoke the simultaneous input/output mode of the `.rd` request.
`-r`*xn*	Set register *x* (one character) to *n.*
`-s`*n*	Stop every *n* pages. The `nroff` formatter will halt after every *n* pages (default *n* = 1) to allow paper loading or changing and will resume upon receipt of a new line. When using `troff`, it is probably preferable to use the `-s` option on the postprocessor if one exists.

Table 3-1 Options for invoking `nroff`/`troff` (*continued*)

Option	Effect
`-T`*name*	Specify the name of the output terminal type. Currently defined names are `lp` for generic printers that can underline and tab, `2631` for the Hewlett-Packard 2631 printer in regular mode, `2631-c` for the Hewlett-Packard 2631 printer in compressed mode, `2631-e` for the Hewlett-Packard 2631 printer in expanded mode, `300` for the DASI 300, `300-12` for DASI 300 terminal set to 12 pitch, `300s` for the DASI 300s, `300s-12` for DASI 300s terminal set to 12 pitch, `37` for the Teletype Model 37 (`nroff` default), `382` for the DCT-382 terminal, `4000a` for the Trendata 4000A terminal, `450` for the DASI 450, `450-12` for the DASI 450 set to 12 pitch, `832` for the Anderson Jacobson 832 terminal, `8510` for the C.ITOH printer, `tn300` for the GE TermiNet 300 (or any terminal without half-line capabilities), and `X` for the EBCDIC TX train printer.
	In `troff`, the `-T` option may be used to specify the output device. The `psc` argument ("`troff -Tpsc`") is required for PostScript output on a LaserWriter. (This is the A/UX `troff` default.)
`-u`[*n*]	(`nroff` only.) Set the emboldening factor (number of character overstrikes) in the formatter for the third font position (bold) to be *n* (0 if *n* is missing). It is not possible to turn off the emboldening in `nroff` if the overstriking is controlled locally by the printing device.
`-z`	Suppress formatted output. Only message output will occur (from `.tm` requests and diagnostics).

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

```
nroff -o4,8-10 -T300s -mabc chapter1 chapter2
```

requests formatting of pages 4, 8, 9, and 10 of a document contained in the files named `chapter1` and `chapter2`, 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`.

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

- 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`.

`troff` output can be viewed on the Teletype Model 5620. No special filter is required to postprocess `troff`'s output for the 5620. The finished version of a document typeset with `troff` is most frequently sent to a phototypesetter:

`tbl` *file* `| eqn | troff [`*options*`] |` *typesetter*

The first pipe (|) indicates the piping of `tbl` output to `eqn` input; the second pipe indicates the piping of `eqn` output to the `troff` formatter input. Finally, the accumulated output from these processes is piped to a postprocessor that interprets `troff`'s output language for the output device.

`tc` is a phototypesetter-simulator postprocessor, which enables you to view `troff` output on a Tektronix 4014 terminal. The syntax for its usage is as follows:

`pic` *file* `| tbl | eqn | troff [`*options*`] | tc`

The `troff` formatter depends on a postprocessor to convert its output into codes for a particular phototypesetter.

Principles of `nroff` and `troff` formatters

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

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 one- or two- character name that specifies a basic request or the substitution of a user-defined macro in place of the control line. The 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, tabs, or both) for aesthetic reasons. Names must be followed by either a space or a newline character. Control lines with unrecognized request/macro names are ignored. The tables throughout this chapter contain explanations of the request/macro names.

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 two-character register name. The escape sequences for characters, indicators, and functions are summarized at the end of this chapter.

Formatter and device resolution

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. Units in `troff` are device-dependent. `troff` rounds horizontal/vertical numeric parameter input to its internal horizontal/vertical resolution. `nroff` similarly rounds numeric input to the actual resolution of the output device indicated by the `-T` option (default Teletype Model 37).

Numeric parameter input

Both `nroff` and `troff` formatters accept numeric input with the appended scale indicators shown in Table 3-2, 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. The number of basic units is device-dependent in `troff`.

Table 3-2 Numeric input and appended scale indicators for `nroff`/`troff`

Scale indicator	Meaning	Number of basic units in `nroff`
i	Inch	240
c	Centimeter	240x50/127
P	Pica = 1/6 inch	240/6
m	Em = S points	C
n	En = em/2	C; same as em
p	Point = 1/72 inch	240/72
u	Basic unit	1
v	Vertical line space	V
None	Default	None

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 requests `.VS` and `.vs` and functions `\H` and `\s`
- *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 numeric 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 prefixed to a number *n* to generate the distance to the vertical or horizontal place *n*:

- For vertically oriented requests and functions, | *n* becomes the distance in basic units from the current vertical place on the page or in a **diversion** to the vertical place *n* (see "Creating Diversions: Storing and Redirecting Text" and "Using Traps" later in this chapter) .
- 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.

Numeric expressions

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

- an expression involving parentheses
- the arithmetic operators +, –, /, *, and % (mod)
- the logical operators <, >, <=, >=, =, ==, & (and), 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 scalings differ. For example, if the number register x contains 2 and the current point size is 10, then

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

sets the line length to 1/2 the sum of 4.25 inches + 2 picas + 3 ems (30 points because the point size is 10).

◆ **Note** The use of white space in arithmetic expressions is not permitted. There is no precedence among arithmetic and logical operators. `nroff/troff` expressions do not recognize decimal multipliers or divisors; a high level of precision may be achieved by mixing scales within expressions. ◆

Notation

Numeric parameters are indicated in this chapter in two ways. A ±n means that the argument may take one of the forms n, +n, and –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, numeric 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 value.

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

`troff` character set

The `troff` character set consists of the so-called Commercial II character set plus the Special Mathematical font character set. The ASCII characters are entered as themselves (with three exceptions); non-ASCII characters are entered in the form \ (*xx*, where *xx* is a two-character name. The three ASCII character exceptions are mapped in Table 3-3.

Table 3-3 ASCII character exceptions to `troff`

ASCII input character	Name	Printed by `troff` character	Name
´	Acute accent	'	Close quotation mark
`	Grave accent	'	Open quotation mark
–	Minus	-	Hyphen

The characters ´, `, and - may be entered by typing \ ´, \ `, and \ -, respectively, or by typing their names (\ (aa, \ (ga, and \ (mi). The ASCII characters @, #, ", ´, `, <, >, \, {, },~, ^, and _ exist in 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 that are available on the output device
- characters that can be constructed by overstriking or by other combinations
- characters that can be mapped into other printable characters

Each printer's capability is determined by a driving table prepared for that device. The characters ´, `, and - print as themselves.

Definitions of terms

Formatter refers to the `nroff` and `troff` text formatting programs. `nroff` and `troff` behave similarly, except where noted.

Requests are built-in commands recognized by the formatters. Although you seldom need to use these requests directly, this chapter refers to some of them. These requests have lowercase names. mm and ms macros have uppercase names, and me macros have lowercase names (for example, .sp is a formatter request, .lp is an me macro, and .PP is an ms macro).

Macros are named collections of requests. The macro name is used as an abbreviation for a collection of commands that you would otherwise have to enter explicitly each time they were used. mm, ms, and me supply many macros, and you can define additional ones. Macros and requests share the same set of names and are used in the same way. Table 5-53 at the end of Chapter 5 lists the ms macros alphabetically, and Table 6-24 at the end of Chapter 6 lists the me macros alphabetically.

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. You can define a string with the .ds (define string) command, and call it out in the form *x (for one-character names) or *(xx (for two-character names). For instance, the string DY in ms contains the current date. The input line

```
Today is \*(DY.
```

prints

Today is October 17, 1989.

You can replace the current date with the command

```
.ds DY 02/21/90
```

Table 5-55 at the end of Chapter 5 lists the ms string names alphabetically.

Number registers are integer variables. These registers are used for flags and for arithmetic and automatic numbering. You can give a register a value with the .nr command. For example, the following sets the value of the line length register, LL:

```
.nr LL 4i
```

This instructs the formatter to generate all text lines at 4 inches. To reset this value to the default, enter the following:

```
.nr LL 0
```

See the section "Extending and Modifying Memorandum Macros" in Chapter 4 for naming conventions for requests, macros, strings, and number registers.

Working with text

The `troff` and `nroff` formatters allow you to choose the font and size you want, overstrike or underline characters, create brackets, and set vertical spacing to meet very specific requirements.

Choosing a font

Default fonts may differ from device to device. Typically, the fonts will include at least the following: Times Roman (R), Times Italic (I), Times Bold (B), and Special Mathematical (S). The current font may be changed by use of the `.ft` request or by embedding at any desired point either \f$x$, \f (xx, or \f$n$, where x and xx are the names of mounted fonts, and n is a numeric font position. It is not necessary to change to the Special Mathematical font; characters on that font are automatically handled. They are invoked by their four-character input names (see "`troff` Character Set" earlier in this chapter).

A request for a named but not mounted font is translated into a request to mount the font at position 0. This position is reserved for such dynamic requests and is otherwise inaccessible. The `troff` processor can be informed that any particular font is mounted by use of the `.fp` request. The list of known fonts is device-dependent. In the subsequent discussion of font-related requests, f represents either a one- or two-character font name or the numeric font position. The current font is available as a numeric position in the read-only number register `.f`.

Font control is understood by the `nroff` formatter, which normally underlines italic characters and overstrikes bold characters. Other font changes are usually ignored.

Setting character size

The available character point sizes depend on the individual printing device. The `.ps` request is used to change or to restore the default point size. Alternatively, the point size may be changed between any two characters by embedding a \s$n$ at the desired point to set the size to n or a \s±n ($1 \leq n \leq 9$) to increase or decrease the size by n; \s0 restores the previous size. Requested point size values that are between two valid sizes yield the closer legal size. The current size is available in the `.s` number register.

In `troff` the escape sequence `\H' n'` sets the height of a character without affecting its width. *n* can be expressed in absolute values or in relative values of the form ±*n*. Note that the `nroff` formatter ignores type size control.

Table 3-4 Character size request forms

Request form	Initial value	If no argument	Explanation
`.bd` *f* [*n*]	Off	–	Boldface font *f* by *n*–1 units. Characters in font *f* will be artificially boldfaced 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 boldface mode is turned off. The mode must still (or again) be in effect when the characters are physically printed.
`.bd` s *f n*	Off	–	Boldface 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.
`.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.
`.fp` *n f* [*file*]	R,I,B,S	Ignored	Position font. A font named *f* is mounted on position *n*. It is a fatal error if *f* is not known. `.fp` accepts a third optional argument, *file*, which is an alternate version of the font *f*.
`.ft` [*f*]	Roman	Previous	Change to font *f* (*f* is *x*, xx, *n*, or P). Font P means the previous font. For font changes within a line of text, sequences `\f x`, `\f` (*xx*, and `\f n` can be used. Relevant parameters are a part of the current environment.
`.ps` [±*n*]	10 point	Previous	Set point size to ±*n*. Any valid positive size value may be requested; if invalid, the nearest valid size will result, with a maximum size to be determined by the individual printing device. A paired sequence +*n*, -*n* will work because the previous requested value is remembered. For point size changes within a line of text, sequence `\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.

.bd can be used to boldface characters, effectively increasing the number of available fonts. This capability of modifying existing fonts to make new ones is enhanced with the troff escape sequence, \s, used to slant output characters by a number of specified degrees. This escape sequence is stated as \s'*n*', where *n* may be any integer, negative or positive. 0 turns slanting off.

Overstriking characters

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. *String* should not contain local vertical motion.

Setting zero-width characters

The function \z*c* will generate *c* without spacing over it and can be used to produce left-aligned overstruck combinations.

Creating 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'* can 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 space, 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

$$\left[\, E \,\right]$$

Underlining

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 the `.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.

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 can be insulated from *n* with a `\&`.

- If *c* is not specified, the base-line rule (_) (underline character in `nroff`) is used .

- 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 a macro can 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

<u>underlined words</u>

and

$\boxed{\text{words in a box}}$

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 underrule were designed to form corners when using one-em vertical spacings. For example, the macro

```
.de eb
.sp -1i        \"compensate for 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 is saved in number register a (for example, using `.mk a`).

In addition, `troff` provides drawing functions capable of drawing arcs and splines; these functions are listed in Table 3-5.

Table 3-5 Line-drawing requests

Request form	Explanation
\D' l *dh dv'*	Draw a line for the current position by *dh, dv*.
\D' c *d'*	Draw a circle of diameter *d* with its left side at the current position.
\D' e *d1 d2'*	Draw an ellipse of diameters *d1* and *d2* with its left side at the current position.
\D' na *dh1 dv1 dh2 dv2'*	Draw a counterclockwise arc from the current position to *dh1+dh2, dv1+dv2,* with its center at *dh1, dv1* from the current position.
\D' ~ *dh1 dv1 dh2 dv2...'*	Draw a B-spline from the current position by *dh1, dv1,* then by *dh2, dv2,* then ...

The current position after using these drawing functions is at the end of the drawn line, which for circles and ellipses is at the right side.

Setting vertical spacing

Vertical spacing size (*v*) between base lines of successive output lines can be set using the .vs request with a device-dependent resolution. 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 (for example, double-spacing) may be obtained with a .ls (line spacing) request.

Adding an extra line space

If a word contains a vertically tall construct requiring the output line containing it to have extra vertical space before or after it or in both places, the extra line space function $\backslash x'n'$ can be embedded in or attached to that word. In this and in other functions having a pair of delimiters around their parameters, 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 used postline extra line space is available in the `.a` register.

Creating a block of vertical space

A block of vertical space is ordinarily requested using `.sp`, which honors the no-space mode and does not space past a trap. A contiguous block of vertical space may be reserved using the `.sv` request. Forms that may be used to request vertical space are listed in Table 3-6.

◆ **Note** Values separated by a semicolon (;) in the "Initial value" field in Table 3-6 are for the `nroff` and `troff` formatters, respectively. ◆

Table 3-6 Vertical space requests

Request form	Initial value	If no argument	Explanation
.ls [*n*]	*n* = 1	Previous	Set line spacing to ±*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.
.ns	Space	—	Set no-space mode, which 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	—	—	Save output 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 of 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 produced. If the distance to the next trap is less than *n*, no vertical space is immediately produced, 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*]	1/6 in. 12pt.	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	—	—	Cause a break and output of a blank line (just as does .sp 1).

Structuring the page

Top and bottom margins are not automatically provided. They may be defined by two macros that set traps at vertical positions 0 (top) and -n (n from the bottom) (see "Using Traps" later in this chapter). 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. 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). Page control request forms are listed in Table 3-7.

Physical limitations on the `nroff` and `troff` processor output are output-device dependent.

◆ **Note** Values separated by a semicolon (;) in the "Initial value" field in Table 3-7 are for the `nroff` and `troff` formatters, respectively. ◆

Table 3-7 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.

Table 3-7 Page control requests (*continued*)

Request form	Initial value	If no argument	Explanation
.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*]	11 in.	11 in.	Set page length 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.
.pn ±*n*	*n* = 1	Ignored	Set 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; 1 in.	Previous	Set 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. 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, for example, using the sequence .mk *r*. . . . sp | \\ \n*r*u. Mode or relevant parameters are associated with current diversion level. The scale indicator is ignored if not specified in the request.

Filling, adjusting, and centering text

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 or the 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 using a backslash-space character (\SPACE); this separates the words with an unpaddable space. 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 flag 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 a period (.), a question mark (?), or an exclamation mark (!) 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 embedded 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.

Controlling line and word breaks

Copying 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. (See Table 3-8.)

Table 3-8 Interrupted text 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:

Indicator	Adjust type
l	Adjust left margin only
r	Adjust right margin only
c	Center
b or n	Adjust both margins
absent	Unchanged

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

Request form	Initial value	If no argument	Explanation
.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 (minus indent). If n = 0, any residual count is cleared. A break occurs after each of the n input lines. If the input line is too long, it will be left-adjusted. The request normally causes a break. Relevant parameters are a part of the current environment.
.fi	Fill	—	Set 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	—	Set 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.
.nf	Fill	—	Set 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.

Hyphenating text

The automatic hyphenation may be switched off and on. When switched on with .hy, several variants may be set. A hyphenation indicator character may be embedded in a word to specify desired hyphenation points or may precede a word 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 entered 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. (See Table 3-9.)

Table 3-9 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 embedded minus signs. Versions of a word with terminal s are implied; that is, *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; that is, $n = 14$ 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.

Indenting lines

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. (See Table 3-10.)

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 produced. In fill mode, the length of text on an output line is less than or equal to the line length minus the indent.

The current line length and indent are available in registers `.l` and `.i`, respectively. The length of three-part titles produced by `.tl` is independently set by `.lt` (see "Creating Three-Part Titles" later in this chapter).

Table 3-10 Line length and indent requests

Request form	Initial value	If no argument	Explanation
`.in` [±n]	$n = 0$	Previous	Indent. The indent is set to ±n and prefixed 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. 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 value of the current indent (stored in the `.i` register) is unchanged. 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.

Setting tabs

Both the ASCII horizontal tab character and the ASCII SOH character (the leader) can 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 Table 3-11

- *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 3-11 Three types of 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 prefixed 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 an uninterpreted tab and leader, respectively, and are equivalent to actual tabs and leaders in copy mode.

Setting field delimiters

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.

◆ **Note** Values separated by a semicolon (;) in the "Initial value" field in Table 3-12 are for the nroff and troff formatters, respectively. ◆

Table 3-12 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:

Type	Result
R	Right
C	Centering
Absent	Left

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

Request form	Initial value	If no argument	Explanation
tc [c]	None	None	Tab repetition character becomes c or is removed specifying motion. Relevant parameters are a part of the current environment.

Advanced features

The following section describes the various advanced features you can use with nroff/troff formatters.

Creating macros and strings

A macro is a named set of arbitrary lines that can be invoked by name or with a trap. A string is a named string of characters, not including a newline character, that can be interpolated by name at any point. Request, macro, and string names share the same name list. Macro and string names may be one- or two-characters long and may usurp previously defined request, macro, or string names. 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 can 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 can be nested within text.

Interpreting copy mode input

During the definition and extension of strings and macros in the current environment, 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 (see "Macros and Strings" earlier in this chapter)

- arguments indicated by \$ are interpolated

- concealed newline characters indicated by \RETURN are eliminated

- comments indicated by \" are eliminated (see "Comments and Concealed Newline Characters")

- \t and \a are interpreted as ASCII horizontal tab and start of heading (SOH), respectively (see "Setting Tabs" later in this chapter)
- \\ is interpreted as "\"
- \. is interpreted as "."

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

Defining arguments

When a macro is invoked by name, the remainder of the line can contain up to nine arguments. The argument separator is the space character, and arguments may be surrounded by double quotation marks to permit embedded space characters. Pairs of double quotation marks may be embedded in double-quoted arguments to represent a single double-quote. If the desired arguments will not fit on a line, a concealed newline character may be used to continue on the next line.

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

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

and called by

```
.xx Monday 14th
```

to produce the text

Today is Monday the 14th.

The \$ was concealed in the definition with a preceding 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
- 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.

Creating diversions: Storing and redirecting text

Processed output may be diverted into a macro for purposes such as footnote processing or determining the horizontal and vertical sizes of some text for conditional changing of pages or columns. A single diversion trap can be set at a specified vertical position. The number registers .dn and .dl, respectively, contain the vertical and horizontal sizes of the most recently ended diversion. Processed text that is diverted into a macro retains the vertical size of each of its lines when reread in no-fill mode regardless of the current v. Constant-spaced (.cs) or emboldened (.bd) text that is diverted can be reread correctly only if these modes are again or still in effect at reread time. One way to do this is to embed in the diversion the appropriate .cs or .bd request with the transparent mechanism (described in "Transparent Throughput" later in this chapter). #

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

- diversion trap and associated macro
- no-space mode
- internally saved marked place (see .mk and .rt)
- current vertical place (.d register)
- current high-water text base line (.h register)
- current diversion name (.z register)

Using traps

Three types of **trap** mechanisms are available:

- page trap
- diversion trap
- input-line-count trap

Macro-invocation traps can be planted using .wh requests at any page position, including the top. This trap position can 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 whose vertical size reaches or sweeps past the trap position is generated. 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, can be planted using .dt requests. The .t register works in a diversion. If there is no subsequent trap, a large distance is returned. (See Table 3-13.)

Table 3-13 Trap 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).
.ch *xx* [*n*]	—	—	Change trap location. Change the trap position for macro *xx* to be *n*. In the absence of *n*, the trap, if it exists, 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.

(continued)➡

Table 3-13 Trap requests (*continued*)

Request form	Initial value	If no argument	Explanation
.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 quotation marks 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.
.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.

Storing values: Creating number registers

A variety of predefined number registers are available to the user. In addition, the user may define his or her own named registers. Register names are one- or two-characters long and do not conflict with request, macro, or string names. Except for certain predefined read-only number registers, a number register can be read, written, automatically incremented or decremented, and interpolated into the input in a variety of formats. One common use of user-defined registers is to automatically number sections, paragraphs, lines, and so on. A number register can be used any time numeric input is expected or desired and can be used in numeric expressions.

Number registers are created and modified using the .nr request, which specifies name, numeric value, and automatic increment size. Registers are also modified if invoked with an automatic incrementing sequence. If the registers x and xx both contain n and have the automatic increment size m, the access sequences have the effects shown in Table 3-14.

Table 3-14 Number register access sequences

Sequence	Effect on reguster	Value interpolated
nx	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 one of the following:

- decimal (default)
- decimal with leading zeros
- lowercase Roman
- uppercase Roman
- lowercase sequential alphabetic
- uppercase sequential alphabetic

The escape sequence "\g*x*" or "\g (*xx*" gives the format used by register *x* or *xx*. This escape sequence will return a value only if the stated register has been set or used; otherwise, it returns 0. The value can also be saved and used as the second argument of .af to restore a previous format. (See Table 3-15.)

Table 3-15 Number register 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
			1 0,1,2,...
			001 000,001,002,...
			i 0,i,ii,...
			I 0,I,II,...
			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.

Creating 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. (See Table 3-16.)

Table 3-16 Three-part title 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 [d	%	Off	Page number character set to c or removed. The page number register remains %.
.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.

Spacing characters on a line: Setting horizontal and vertical motion and width

This section explains how `troff` creates superscripts and subscripts and how you can space characters horizontally on a line by adding or reducing space.

Moving characters within a line: Setting 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 0.

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

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

Spacing characters within a line: Setting width

The **width function** \w' *string'* generates the numeric width of *string* in basic units. Size and font changes may be embedded in *string* and will not affect the current environment. For example,

```
.ti -\w'1.'u
```

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

The width function also sets three number registers. The registers st and sb are sets, respectively, to the highest and lowest extents of *string* relative to the base line; 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.

Overprinting text: Marking 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 boldface *word* by backing up and overprinting it, resulting in

word

Numbering output lines

Automatic sequence numbering of output lines can be requested with .nm. When it is in effect, a three-digit Arabic number and a digit space are prefixed 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* can 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). (See Table 3-17.)

Table 3-17 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 1n. 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.

The following example illustrates output line numbering. Paragraph portions are numbered with $m = 2$.

Automatic sequence numbering of output lines may be
2   requested with .nm. When in effect, a three-digit Arabic number and a digit space areeee prefixed to output text four lines. Text lines are offset by four
4   digit spaces and otherwise retain their line length. A reduction in line length (such as .ll -\w'0000'u in this example) may be desired to keep the right margin aligned with an earlier margin.
6   Blank lines, other vertical spaces, and lines generated 10 by .tl are
8   not numbered. Numbering can be temporarily suspended with .nn or with a .nm followed by a later .nm 12 +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 example uses the multiple of 2.

■ .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.

Using conditionals

In Table 3-18, which is a summary and explanation of conditional acceptance requests,

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

Table 3-18 Summary and explanation of conditional acceptance 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* is 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* is false, accept *anything*.
.if *n anything*	If expression $n > 0$, accept *anything*. The scale indicator is ignored if not specified in the request.
.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 3-19 lists built-in condition names.

Table 3-19 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 0, 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 %'''
```

generates 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.

Switching environments

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. (See Table 3-20.)

Table 3-20 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.

Inserting 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 A/UX 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 flag 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 can 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. (See Table 3-21.)

Table 3-21 Standard input insertion 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*.

Switching input/output files

Table 3-22 lists requests for switching input/output files.

Table 3-22 Input/output switching requests

Request form	Initial value	If no argument	Explanation
.cf *filename*	—	—	Copy the contents of *file*, uninterpreted into `troff` output file at this point. Havoc ensues unless the motions in the file restore the current horizontal and vertical positions.
.lf *n file*	—	—	Correct `troff`'s idea of the current line number, *n*, and the current file, *file*, for use in error messages.
.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*. 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.

Reading output and error messages

Output from `.tm` and `.pm`, prompt from `.rd`, and various error messages are written onto the A/UX 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. (See Table 3-23.)

Various error conditions can 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 usual procedure 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. Error conditions that can cause this include the inability to create, read, or write files, and the exceeding of certain internal limits that make future output unlikely to be useful.

Table 3-23 Output printing request

Request form	Initial value	If no argument	Explanation
`.ab` [*text*]	—	—	Print *text* on the message output and terminate without further processing. If *text* is missing, `User Abort.` is printed. This request does not cause a break. The output buffer is flushed.

Miscellaneous requests

Table 3-24 lists those requests that are not found in other tables, such as requests that flush the output buffer, ignore input lines, set margin character, print macro, execute *cmd* without capturing output, and print *string* on a terminal.

Table 3-24 Miscellaneous requests

Request form	Initial value	If no argument	Explanation
.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	Set margin character *c* and separation n. Specify 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.
.sy *cmd args*	—	—	*cmd* is executed but its output is not captured at this point. The standard input for *cmd* is closed. Output for processing must be explicitly saved in an output file.
.tm [*string*]	—	Newline	Print *string* on terminal (A/UX 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.

Input/output conventions and character translations

The following sections explain input/output characters and conventions found in `nroff`/`troff` formatters.

Input character translations

The newline character delimits input lines. In addition, STX, ETX, ENQ, ACK, and BEL are accepted and can be used as delimiters or translated into a graphic with a `.tr` request. All others are ignored.

The escape character (`\`) introduces sequences that indicate some function such as a font change or the printing of a special character. 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 can be turned off with `.eo` and restored with `.ec`.

Ligatures

Two **ligatures** are available in the `troff` character set: fi and fl. They may be entered (even in the `nroff` formatter) by `\(fi` and `\(fl`, respectively. Note that ligature mode is normally on in the `troff` formatter; that is, ligatures are automatically produced. Constant-width fonts normally do not use ligatures.

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 with any trap-invoked macros.

Output translation

One character can be made a stand-in for another character using the .tr request. All text processing (for example, 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).

◆ **Note** Values separated by a semicolon (;) in the "Initial value" field in Table 3-25 are for the nroff and troff formatters, respectively. ◆

Table 3-25 Output 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 two-character ligatures are automatically invoked. Ligature mode is inhibited for requests, macros, strings, registers, filenames, and copy mode. There is no effect in the nroff formatter.

Table 3-25 Output translation requests (*continued*)

Request form	Initial value	If no argument	Explanation
`.tr` *abcd...*	None	—	Translate *a* into *b*, *c* into *d*, and so forth 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).
`.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.

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 is known as **transparent throughput.** This mechanism may be used to pass control information to a postprocessor or to embed control lines in a macro created by a diversion.

Comments and concealed newline characters

An unusually long input line that must stay one line (for example, 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 \RETURN is ignored except in a **comment.** Comments can be embedded at the end of any line by prefacing them with \ ". The newline character at the end of a comment cannot be concealed. A line beginning with \ " will appear as a blank line and behave like `.sp 1`; a comment can be on a line by itself by beginning the line with `.` \ ".

Reference tables

The following tables are your guides to escape sequences, naming conventions, and predefined number registers.

Table 3-26 Escape sequences for characters, indicators, and functions

Escape sequence	Meaning
\\	\ (to prevent or delay the interpretation of \)
\e	Printable version of current escape character.
\´	Acute accent (equivalent to \ (aa).
\`	Grave accent (equivalent to \ (ga).
\-	− (minus sign in the current font).
\.	Period (dot).
\SPACE BAR	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 ($1 \leq n \leq 9$).
\%	Default optional hyphenation character.
\ (xx	Character named xx.
*x, * (xx	Interpolate string x or xx.
\{	Begin conditional input.
\}	End conditional input.
\RETURN	Concealed (ignored) newline character.
\a	Uninterpreted leader character.
\b' abc...'	Bracket building function.
\c	Continuation of interrupted text.
\d	Forward (down) 1/2-em vertical motion (1/2 line in nroff).
\D	Line-drawing functions.

Table 3-26 Escape sequences for characters, indicators, and functions (*continued*)

Escape sequence	Meaning
\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*.
\h' *n'*	Local horizontal motion, move right *n* (negative left).
\H' *n'*	Height control of characters (does not affect width).
\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	Uninterpreted horizontal tab.
\u	Reverse (up) 1/2-em vertical motion (1/2 line in nroff).
\v' *n'*	Local vertical motion, move down *n* (negative up).
\w' *string'*	Interpolate width of *string*.
\x' n'	Extra line-space function (negative before, positive after).
\zc	Print *c* with zero width (without spacing).
\X	Any character not listed above.

◆ **Note** Escape sequences \\, \., \", \$, *, \a, \n, \t, and \RETURN are interpreted in copy mode. ◆

Table 3-27 Naming conventions for special characters on the standard fonts

Char	Input name	Character name	Char	Input name	Character name
'	´	Close quotation mark	1/2	\(12	One-half
'	`	Open quotation mark	3/4	\(34	Three-fourths
—	\(em	3/4-em dash	fi	\(fi	fi
-	–	Hyphen	fl	\(fl	fl
-	\(hy	Hyphen	°	\(de	Degree
–	\-	Current font minus	†	\(dg	Dagger
•	\(bu	Bullet	¢	\(fm	Foot mark
◻	\(sq	Square	¢	\(ct	Cent sign
_	\(ru	Rule	®	\(rg	Registered
1/4	\(14	One-fourth	©	\(co	Copyright

Table 3-28 Naming conventions for 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	l(*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*	υ	\(*n	nu
Ξ	\(*C	Xi	ξ	\(*c	xi
O	\(*O	Omicron*	o	\(*o	omicron
Π	\(*P	Pi	π	\(*p	pi
P	l(*R	Rho*	ρ	\(*r	rho
Σ	\(*S	Sigma	σ	\(*s	sigma
			ς	\(ts	terminal sigma
T	\(*T	Tau*	τ	\(*t	tau
Ψ	\(*U	Upsilon	0	\(*u	upsilon
Φ	\(*F	Phi	φ	\(*f	phi
X	\(*X	Chi*	0	\(*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 3-29 Naming conventions for special characters 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 arron
~	\(ap	Approximates	↓	\(da	Down arrow
≠	\(!=	Not equal	‡	\(dd	Double dagger
√	\(sr	Square route	♥	\(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 3-30 Predefined general number registers

Register name	Description
%	Current page number.
.b	Boldfacing factor of the current font.
.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.
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).
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.

Table 3-31 Predefined read-only number registers

Register name	Description
.$	Number of arguments available at the current macro level.
$$	Identification number (process ID) for `nroff` or `troff` processes.
.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 0 otherwise, that is, if the current page did not appear in the −o option list.
.T	Set to 1 in the `nroff` formatter if −T flag 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 `.`
.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.

4 mm Macros

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 A/UX text formatters `nroff` and `troff`. For more details, see the previous chapter or refer to `nroff`(1) and `troff`(1) in *A/UX Command Reference*.

What are mm macros, and why should you use them?

The following qualities of mm 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 will 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.

- *Setting parameters*—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 documents that meet their respective needs with 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.

- *Uncoupling 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 can be made to vary from document to document or within a single document.

Required structure for a document

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:

- The parameter-setting segment sets the general style and appearance of a document. The user can control page width, margin justification, numbering styles for headings 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.

- The beginning segment includes those items that occur only once, at the beginning of a document, for example, title, author's name, and date.

- The 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 (see "Creating Numbered Headings" later in this chapter). 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 (see "Creating Lists" later in this chapter). The body may also contain various types of displays, tables, figures, footnotes, and references (see "Creating Displays," "Creating Footnotes," and "Using References" later in this chapter).

- The ending segment contains those items that occur only once at the end of a document. Included are signatures and lists of notations (for example, "Copy to" lists) (see "Creating End-of-Memorandum Macros" later in this chapter). 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 (see "Generating a Table of Contents and Cover Sheet" later in this chapter).

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

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.

Restricted use of the 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, for example, in page headers and footers, headings, and lists. Users who include BEL characters in their input text file (especially in arguments to macros) will receive mangled output. See "Creating Page Headers and Footers," "Creating Paragraphs," "Creating Numbered Headings," and "Creating Lists" later in this chapter.

Options and commands for accessing
mm macros

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

The mm command

The mm(1) command can be used to prepare documents using the nroff formatter and the memorandum macros. The mm command has options to specify preprocessing by tbl or neqn, or both, and for postprocessing by various output filters.

◆ **Note** Options can occur in any order but must appear before the filenames. ◆

Any arguments or flag options that are not recognized by the mm command (for example, -rC3) are passed to the nroff formatter or to mm, as appropriate. Options are shown in Table 4-1.

Table 4-1 mm command options

Option	Meaning
-e	The neqn preprocessor is to be invoked; also causes neqn to read /usr/pub/eqnchar (see eqnchar(7)).
-t	The tbl(1) preprocessor 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.
-12	The 12-pitch mode is to be used. The pitch switch on the terminal should be set to 12 if necessary.
-T2631	Output is prepared for an HP2631 printer, where -T2631-e and -T2631-c may be used for expanded and compressed modes, respectively (implies -c).
-T300	Output is to a DASI 300 terminal.
-T300s	Output is to a DASI 300S.
-T37	Output is to a Teletype Model 37.
-T382	Output is to a DTC-382.
-T4000a	Output is to a Trendata 4000A.
-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.
-T832	Output is to an Anderson Jacobson 832 terminal.
-T8510	Output is to a C.ITOH printer.
-Tlp	Output is to a device with no reverse or partial line motions or other special features (implies -c).
-Ttn300	Output is to a GE TermiNet 300 terminal.
-TX	Output is prepared for an EBCDIC line printer.

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)).

The -mm flag

The mm package can also be invoked by including the -mm flag as an argument to the formatter. 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
- initializes the formatter to be ready to process input text files

Typical command lines

The prototype command lines are as follows (various options are explained in "Parameters Set From the Command Line" later in this chapter):

- Text without tables or equations:

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

  or

  nroff [*options*] -mm *filename* . . .
  mmt [*options*] *filename* . . .

  or

  troff [*options*] -mm *filename* . . .
- Text with tables:

  mm -t [*options*] *filename* . . .

  or

  tbl *filename* . . . | nroff [*options*] -mm
  mmt -t [*options*] *filename* . . .

  or

  tbl *filename* . . . | troff [*options*] -mm

- Text with equations:

  `mm -e [`*options*`] ` *filename* `...`

  or

  `neqn /usr/pub/eqnchar` *filename* `... | nroff [`*options*`] -mm`
  `mmt -e [`*options*`] ` *filename* `...`

  or

  `eqn /usr/pub/eqnchar` *filename* `... | troff [`*options*`] -mm`

- Text with both tables and equations:

  `mm -t -e [`*options*`] ` *filename* `...`

  or

  `tbl` *filename* `... | neqn /usr/pub/eqnchar\`
  `  | nroff [`*options*`] -mm`

  `mmt -t -e [`*options*`] ` *filename* `...`

  or

  `tbl` *filename* `... | eqn /usr/pub/eqnchar\`
  `  | troff [`*options*`] -mm`

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 (for example, 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. For more information, see "Parameters Set From the Command Line" later in this chapter and 300(1), 450(1), 4014(1), hp(1), col(1), termio(4), and term(5).

- 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 -mm` *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 -mm *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 -mm *filename* ...

- Hewlett-Packard HP264x CRT family:

  mm -Thp *filename* ...

  or

  nroff -mm *filename* ... | col | hp

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

  mm -T745 *filename* ...

  or

  nroff -mm *filename* ... | col -x

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

If two-column processing is used with the nroff formatter, either the -c option must be specified to mm (mm uses the col program automatically for many terminal types), or the nroff formatter output must be postprocessed by col. See col(1) in *A/UX Command Reference* and "Creating Two-Column Output" and "The mm Command" in this chapter. 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 (for example, 450(1)); mm(1) with the -c option handles all this automatically.

Parameters set from the 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 shown in Table 4-2.

Table 4-2 Number registers to hold parameter values

Register name	Description
−rA*n*	*n* = 1, has the effect of invoking the .AF macro without an argument (see "Using an Alternate First-Page Format" later in this chapter).
−rC*n*	Sets type of copy (for example, DRAFT) to be printed at the bottom of each page (see "Page Footers" later in this chapter):
	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 the 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 (see "Page Headers" and "SCCS Release Identification" later in this chapter).
−rE*n*	Controls the 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 for inches, v for vertical spaces).
	Default value is 66 lines per page.

Hanging indents with tabs

The following example illustrates the use of the hanging indent feature of variable-item lists (see "Creating a Variable-Item List" earlier in this chapter). 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 two-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 29
.aX .nh off \- no
No hyphenation.
Automatic hyphenation is turned off.
Words containing hyphens
(for example, mother-in-law) can still be
split across lines.
.aX .hy on \- no
Hyphenate.
```

```
Automatic hyphenation is turned on.
.aX .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.
Prefixing the indicator to a word has the
effect of preventing hyphenation of that word.
.LE
```

Note that the space following ".hc\" is required.

The resulting output is

.nh	off	-	no	No hyphenation. Automatic hyphenation is turned off. Words containing hyphens (for example, mother-in-law) may still be split across lines.
.hy	on	-	no	Hyphenate. Automatic hyphenation is turned on.
.hc	c	none	none	Hyphenation indicator character is set to "c" or removed. During text processing, the indicator is suppressed and will not appear in the output. Prefixing the indicator to a word has the effect of preventing hyphenation of that word.

mm examples

This section contains an example of an input file of a simple letter (Figure 4-1) that is also shown formatted by both nroff (in Figure 4-2) and troff (in Figure 4-3) using the memorandum macros. This example illustrates how the formatters work and what to expect from your input file.

Figure 4-1 Example of input file for a simple letter

Input:

```
.nr N 2      \" specifies header to be omitted from page 1
.ta 3i
September 5, 1987
.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
.I
A/UX\*F Text-Processing Tools.
.R
.FS
A/UX is a registered trademark of Apple Computer, Inc.
.FE
.P
This manual covers using the
\s-1UNIX\s+1\*F
.FS
\s-1UNIX\s+1 is a registered trademark of UNIX System
.Laboratories, Inc.
.FE
operating system for preparing documentation, and includes
topics such as:
.VL 17
.LI Formatters:
```

```
the \fBnroff/troff\fR formatters,
with tables listing defaults and explanations of all requests
.LI Tables:
the \fBtbl\fR:program,
with examples of code at the end of the chapter
.LI Equations:
the .fBeqn\fR program, for printing mathematical expressions
.LI "Macro Package:"
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/dcb
Enc.
.fi
```

Figure 4-2 Example of a simple letter: nroff output

                                        September 5, 1987

Mr. Steven J. Jones
386 Broderick Street
San Francisco, CA   94111

Dear Mr. Jones:

Enclosed please find a copy of A/UX[1] Text-Processing Tools.

This manual covers using the UNIX[2] operating system for
preparing documentation, and includes topics such as:

Formatters: the **nroff/troff** formatters, with tables
                 listing defaults and explanations of all
                 requests

Tables: the **tbl** program, with examples of code at the
                 end of the chapter

Equations: the **eqn** program, for printing mathematical
                 expressions

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/dcb
Enc.

_____

[1] A/UX is a registered trademark of Apple Computer, Inc.

[2] UNIX is a registered trademark of UNIX System Laboratories, Inc.

Figure 4-3 Example of a simple letter: `troff` output

September 5, 1987

Mr. Steven J. Jones
386 Broderick Street
San Francisco, CA 94111

Dear Mr. Jones:

Enclosed please find a copy of *A/UX[1] Text-Processing Tools*.

This manual covers using the UNIX[2] operating system for preparing documentation, and includes topics such as:

Formatters: the **nroff/troff** formatters, with tables listing defaults and
                     explanations of all requests

Tables: the **tbl** program, with examples of code at the end of the chapter

Equations: the **eqn** program, for printing mathematical expressions

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/dcb
Enc.

---

[1] A/UX is a registered trademark of Apple Computer, Inc.

[2] UNIX is a registered trademark of UNIX System Laboratories, Inc.

mm reference tables

Tables 4-15 through 4-19 are useful reference tools when using the memorandum macros. Table 4-15 is an alphabetic summary of all the memorandum macro names available for producing a document. Table 4-16 is a summary of all the predefined string names in the memorandum macro package. Table 4-17 is a summary of all the predefined number register names in the memorandum macro package. Tables 4-18 and 4-19 list error messages that you may encounter when formatting a document. memorandum macro error messages as well as `nroff/troff` error messages are explained.

Table 4-15 Memorandum macro names

Macro	Description
1C	One-column processing `.1C`
2C	Two-column processing `.2C`
AE	Abstract end `.AE`
AF	Alternate format of "*Subject/Date/From*" block `.AF` [*company-name*]
AL	Automatically incremented list start `.AL` [*type*] [*text-indent*] [1]
AS	Abstract start `.AS` [*arg*] [*indent*]
AT	Author's title `.AT` [*title*] . . .
AU	Author information `.AU` *name* [*initials*] [*loc*] [*dept*] [*ext*] [*room*] [*arg*] [*arg*] [*arg*]
AV	Approval signature `.AV` [*name*]
B	Bold `.B` [*bold-arg*] [*prev-font-arg*] [*bold*] [*prev*] [*bold*] [*prev*]

Table 4-15 Memorandum macro names (*continued*)

Macro	Description
BE	Bottom block end `.BE`
BI	Bold/italic `.BI` [*bold-arg*] [*italic-arg*] [*bold*] [*italic*] [*bold*] [*italic*]
BL	Bulleted list start `.BL` [*text-indent*] [1]
BR	Bold/roman `.BR` [*bold-arg*] [*roman-arg*] [*bold*] [*roman*] [*bold*] [*roman*]
BS	Bottom block start `.BS`
CS	Cover sheet `.CS` [*pages*] [*other*] [*total*] [*figs*] [*tbls*] [*refs*]
DE	Display end `.DE`
DF	Display floating start `.DF` [*format*] [*fill*] [*right-indent*]
DL	Dashed list start `.DL` [*text-indent*] [1]
DS	Display static start `.DS` [*format*] [*fill*] [*right-indent*]
EC	Equation caption `.EC` [*title*] [*override*] [*flag*]
EF	Even-page footer `.EF` [*arg*]
EH	Even-page header `.EH` [*arg*]
EN	End equation display `.EN`
EQ	Equation display start `.EQ` [*label*]

(continued)➡

Table 4-15 Memorandum macro names (*continued*)

Macro	Description
EX	Exhibit caption .EX [*title*] [*override*] [*flag*]
FC	Formal closing .FC [*closing*]
FD	Footnote default format .FD [*arg*] [1]
FE	Footnote end .FE
FG	Figure title .FG [*title*] [*override*] [*flag*]
FS	Footnote start .FS [*label*]
H	Heading—numbered .H *level* [*heading-text*] [*heading-suffix*]
HC	Hyphenation character .HC [*hyphenation-indicator*]
HM	Heading mark style (Arabic or Roman numerals, or letters) .HM [*arg1*] . . . [*arg7*]
HU	Heading—unnumbered .HU *heading-text*
HX*	Heading user exit X (before printing heading) .HX *dlevel rlevel heading-text*
HY*	Heading user exit Y (before printing heading) .HY *dlevel rlevel heading-text*
HZ*	Heading user exit Z (after printing heading) .HZ *dlevel rlevel heading-text*
I	Italic (underline in the nroff formatter) .I [*italic-arg*] [*prev-font-arg*] [*italic*] [*prev*] [*italic*] [*prev*]
IA	Inside address start .IA [*addressee-name*] [*title*]
IB	Italic/bold .IB [*italic-arg*] [*bold-arg*] [*italic*] [*bold*] [*italic*] [*bold*]

Table 4-15 Memorandum macro names (*continued*)

Macro	Description
IE	Inside address end .IE
IR	Italic/roman .IR [*italic-arg*] [*roman-arg*] [*italic*] [*roman*] [*italic*] [*roman*]
LB	List begin LB *text-indent mark-indent pad type* [*mark*] [*LI-space*] [*LB-space*]
LC	List-status clear .LC [*list-level*]
LE	List end .LE [1]
LI	List item .LI [*mark*] [1]
LO	Letter options .LO *type* [*arg*]
LT	Letter type .LT [*arg*]
ML	Marked list start .ML *mark* [*text-indent*] [1]
MT	Memorandum type .MT [*type*] [*addressee*] or .MT 4 1
ND	New date .ND *new-date*
NE	Notation end .NE
nP	Double-line indented paragraphs .nP
NS	Notation start .NS [*arg*]
OF	Odd-page footer .OF [*arg*]

(continued)➡

Table 4-2 Number registers to hold parameter values (*continued*)

Register name	Description
`-rN`*n*	Specifies page numbering style:

n = 0 (default), all pages get the prevailing header.

n = 1, page header replaces footer on page 1 only.

n = 2, page header is omitted from page 1.

n = 3, "section-page" numbering occurs (`.FD` and `.RP` define footnote and reference numbering in sections). (See "Page Headers," "Using Headings in Page Numbering," "Controlling Format Style of Footnote Text," and "Generating a Reference Page" later in this chapter.)

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 (see "Page Headers" and "Page Footers" later in this chapter), the value of N is irrelevant.

Register name	Description
`-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 in `nroff` and 0.5 inch in `troff`.

Register name	Description
`-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.

Register name	Description
`-rS`*n*	Sets point size and vertical spacing for the document. The default *n* is 10, that is, 10-point type on 12-point vertical spacing, giving 6 lines per inch (see "Setting Point Size and Vertical Spacing" later in this chapter).

This flag applies to the `troff` formatter only.

(continued)➡

Table 4-15 Memorandum macro names (*continued*)

Macro	Description
OH	Odd-page header .OH [*arg*]
OK	Other keywords for technical memo cover sheet .OK [*keyword*] . . .
OP	Odd page .OP
P	Paragraph .P [*type*]
PF	Page footer .PF [*arg*]
PH	Page header .PH [*arg*]
PM	Proprietary marking .PM [*code*]
PX*	Page-header user exit .PX
R	Return to regular (roman) font .R
RB	Roman/bold .RB [*roman-arg*] [*bold-arg*] [*roman*] [*bold*] [*roman*] [*bold*]
RD	Read insertion from terminal .RD [*prompt*] [*diversion*] [*string*]
RF	Reference end .RF
RI	Roman/italic .RI [*roman-arg*] [*italic-arg*] [*roman*] [*italic*] [*roman*] [*italic*]
RL	Reference list start .RL [*text-indent*] [1]
RP	Produce reference page .RP [*arg*] [*arg*]
RS	Reference start .RS [*string-name*]

Table 4-15 Memorandum macro names (*continued*)

Macro	Description
S	Set troff formatter point size and vertical spacing . S [*size*] [*spacing*]
SA	Set adjustment (right-margin justification) default . SA [*arg*]
SG	Signature line . SG [*arg*] [1]
SK	Skip pages . SK [*pages*]
SM	Make a string smaller . SM *string1* [*string2*] [*string3*]
SP	Space vertically . SP [*lines*]
TB	Table title . TB [*title*] [*override*] [*flag*]
TC	Table of contents . TC [*slevel*] [*spacing*] [*tlevel*] [*tab*] [*head1*] [*head2*] [*head3*] [*head4*] [*head5*]
TE	Table end . TE
TH	Table header . TH [N]
TL	Title of memorandum . TL [*charging-case*] [*filing-case*]
TM	Technical memorandum number(s) . TM [*number*] . . .
TP*	Top-of-page macro . TP
TS	Table start . TS [H]
TX*	Table of contents user exit . TX

(continued)➡

Table 4-15 Memorandum macro names (*continued*)

Macro	Description
TY*	Table of contents user exit (suppress CONTENTS) `.TY`
VL	Variable-item list start `.VL` *text-indent* [*mark-indent*] [1]
VM	Vertical margins `.VM` [*top*] [*bottom*]
WA	Writer's address start `.WA` *writer-name* [*title*]
WC	Footnote and display width control `.WC` [*format*]

* Macros marked with an asterisk are not, in general, called directly by the user. They are "user exits" defined by the user and called by mm from inside header, footer, or other macros.

Table 4-16 String names

String	Description
BU	Bullet (`nroff` overstrikes a 0 with a plus sign; `troff` types a filled bullet).
Ci	Table of contents indent list; up to seven scaled arguments for heading levels.
DT	Date (current date, unless overridden); month, day, year (for example, May 1, 1988).
EM	Em-dash string; produces an em dash in the `troff` formatter and a double hyphen in `nroff`.

F	Footnote number generator.
	`nroff: \u\\n+(:p\d`
	`troff: \v'-.4m'\s-3\\n+(:p\s0\v'.4m'`

HF	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	Heading point size list; up to seven codes for heading levels 1 through 7.
Le	Title for list of equations.
Lf	Title for list of figures.
Lt	Title for list of tables.
Lx	Title for list of exhibits.

Table 4-16 String names (*continued*)

String	Description
RE	SCCS release and level of memorandum macros release level (for example, 15.129).
Rf	Reference number generator.
Rp	Title for references .
Tm	Trademark string; places "TM" 1/2 line above text that it follows; seven accent strings are also available.

◆ **Note** 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. ◆

Table 4-17 Number register names

Register	Description
A*	Handles preprinted forms and Bell System logo 0, [0:2]
Au	Inhibits printing of author information 1, [0:1]
C*	Copy type (original, draft, etc.) 0 (original), [0:4]
Cl	Level of headings saved for table of contents 2, [0:7]
Cp	Placement of list of figures, etc. 1 (on separate pages), [0:1]
D*	Debug flag 0, [0:1]
De	Display eject register for floating displays 0, [0:1]
Df	Display format register for floating displays 5, [0:5]

Table 4-17 Number register names (*continued*)

Register	Description
Ds	Static display pre and postspace 1, [0:1]
E*	Controls font of the Subject/Date/From fields 1 (nroff), 0 (troff), [0:1]
Ec	Equation counter, used by .EC macro 0, [0:?], incremented by 1 for each .EC call
Ej	Page-ejection flag for headings 0 (no eject), [0:7]
Eq	Equation label placement 0 (right-adjusted), [0:1]
Ex	Exhibit counter, used by .EX macro 0, [0:?], incremented by 1 for each .EX call
Fg	Figure counter, used by .FG macro 0, [0:?], incremented by 1 for each .FG call
Fs	Footnote space (i.e., spacing between footnotes) 1, [0:?]
H1-H7	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	Heading break level (after .H and .HU) 2, [0:7]
Hc	Heading centering level for .H and .HU 0 (no centered headings), [0:7]
Hi	Heading temporary indent (after .H and .HU) 1 (indent as paragraph), [0:2]
Hs	Heading space level (after .H and .HU) 2 (space only after .H 1 and .H 2), [0:7]
Ht	Heading type (for .H: single or concatenated numbers) 0 (concatenated numbers: 1.1.1, etc.), [0:1]
Hu	Heading level for unnumbered heading (.HU) 2 (.HU at the same level as .H 2), [0:7]

Table 4-17 Number register names (*continued*)

Register	Description
Hy	Hyphenation control for body of document 0 (automatic hyphenation off), [0:1]
L*	Length of page 66, [20:?] (11i, [2i:?] in troff formatter)
Le	List of equations 0 (list not produced), [0:1]
Lf	List of figures 1 (list produced), [0:1]
Li	List indent 6 (nroff), 5 (troff), [0:?]
Ls	List spacing between items by level 6 (spacing between all levels), [0:6]
Lt	List of tables 1 (list produced), [0:1]
Lx	List of exhibits 1 (list produced), [0:1]
N*	Numbering style 0, [0:5]
Np	Numbering style for paragraphs 0 (unnumbered), [0:1]
O*	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	Table of contents page numbering style 0 (lowercase Roman), [0:1]
Of	Figure caption style 0 (period separator), [0:1]
P-	Page number managed by memorandum macros 0, [0:?]
Pi	Paragraph indent 5 (nroff), 3 (troff), [0:?]

(continued)➡

Table 4-17 Number register names (*continued*)

Register	Description
P s	Paragraph spacing 1 (one blank space between paragraphs), [0:?]
Pt	Paragraph type 0 (paragraphs always left justified), [0:2]
Pv	"PRIVATE" header 0 (not printed), [0:2]
Rf	Reference counter, used by . RS macro 0, [0:?], incremented by 1 for each . RS call
S*	t r o f f formatter default point size 10, [6:36]
Si	Standard indent for displays 5 (nroff), 3 (troff), [0:?]
T*	Type of nroff output device 0, [0:2]
Tb	Table counter, used by . TB macro 0, [0:?], incremented by 1 for each . TB call
U*	Underlining style (nroff) for . H and . HU 0 (continuous underline when possible), [0:1]
W*	Width of page (line and title length) 6i, [10:1365] (6i, [2i:7.54i] in the t r o f f formatter)

* Register names marked with an asterisk can be set only from the command line or before the macro definitions are read by the formatter.

Error messages

The following sections list mm error messages and formatter error messages.

mm 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. The error messages are listed in Table 4-18 in alphabetical order by macro name, each with a more complete explanation.

Table 4-18 mm error messages

Error message	Description
Check TL, AU, AS, sequence	Something has disturbed the correct order of macros at the AE, MT start of a memorandum . See "Understanding the Sequence of Beginning Letter Macros" earlier in this chapter.
Check TL, AU, AS, AE, NS, NE, MT sequence	Occurs if the .AS 2 macro was used. Something has disturbed the correct order of macros at the start of a memorandum . See "Understanding the Sequence of Beginning Macros" earlier in this chapter.
Check WA, WE, IA, IE, LT sequence	Something has disturbed the correct order of these macros.
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.
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; that is, 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; that is, have been accumulated but not yet output.
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, that is, 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.

(continued)➡

Table 4-18 mm error messages (*continued*)

Error message	Description
`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; that is, a `.DS` or `.DF` occurs without a matching `.DE`.
`FS:missing FE`	A previous `.FS` macro was not matched by a closing `.FE`; that is, 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 1 to 7, 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. This is not a fatal error. The message is issued because some problem exists in the preceding text.
`LI:no lists active`	The `.LI` macro occurred without a preceding list-initialization macro. The latter probably has been omitted or entered incorrectly.
`LO:LO argument not recognized`	You have provided an argument to `.LO` that it does not recognize.
`LT:LT argument not recognized`	You have provided an argument to `.LT` that it does not recognize.
`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 `.RF` to match it.
`RP: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.

Table 4-18 mm error messages (*continued*)

Error message	Description
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.
)W:WA macro missing	If you use .LT, you must specify at least one .WA/.WE pair.
)W:WA or WE macro missing	If you use .WA or .WE, you must specify the other member of the missing macro pair.
)W:WA, WE, or IE macro missing	You have omitted either or both of the .IA and .IE macros.
WC:unknown option	An incorrect argument has been given to the .WC macro.

Formatter error messages

Most messages issued by the formatter are self-explanatory. Those error messages over which the user has some control are listed in Table 4-19. Any other error messages should be reported to the local system support group.

Table 4-19 Formatter error messages

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 (for example, 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

Table 4-2 Number registers to hold parameter values (*continued*)

Register name	Description
-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 (see "Emphasizing Headings with Bold, Italics, and Underlining" later in this chapter).
	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).

Omission of -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 "Parameters Set From Command Line"

```
.so /usr/lib/tmac/tmac.m
```

remainder of text

In this case, the user must not use the -mm flag (or the mm(1) or mmt(1) command); the .so request has the equivalent effect, but registers shown in "Parameters Set From the Command Line" earlier in this chapter 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,

Table 4-19 Formatter error messages (*continued*)

Error message	Description
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 type;	Indicates a request has been made to mount an unknown font `font`
Nonexistent macro file;	Indicates the requested macro package does not exist.
Nonexistent type;	Indicates the terminal options refer to an unknown terminal `terminal` type.
Out of temp file space;	Indicates additional temporary space for macro definitions, diversions, and so on cannot be allocated. This message often occurs because of unclosed diversions (missing `.FE` or `.DE`), unclosed macro definitions (for example, missing `".."`), or a huge table of contents.
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 page numbers;	Indicates the list of pages specified to the `-o` formatter option is too long.
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)`.

```
.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.

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 (see "Parameters Set From the Command Line" earlier in this chapter) on the command line. This causes the RE string to be generated as part of the page header (see "Page Headers" later in this chapter).

Working with text

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 as necessary (see "Hyphenating Text"). It is possible to turn any of these modes on and off by using .SA (see "Justifying the Right Margin"), Hy (see "Hyphenating Text"), and the .nf and .fi formatter requests. Turning off fill mode also turns off justification and hyphenation.

Understanding formatting

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 can be used with mm (see "Using Formatter Requests" later in this chapter); 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) the overall style of the document

- complicated features such as footnotes or tables of contents can be obtained with ease

- the user is insulated from the complexities of the formatter language

Using arguments and double quotation marks

For any macro call, a null argument is an argument whose width is 0. 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 (for example, the .MT macro; see "Understanding Memorandum Types" later in this chapter). 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 quotation marks. A double quotation mark (") is a single character that should not be confused with two close quotation marks (' ') or open quotation marks (` `). Unless you enclose an argument containing spaces in double quotation marks, it will be treated as several separate arguments.

Double quotation marks 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 close quotation marks (' ') or open quotation marks (` `) or a combination of the two 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.

Specifying 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, that is, an **unpaddable space.** There are several ways to accomplish this:

- Type a backslash followed by a space. This pair of characters directly generates an unpaddable space.

- Sacrifice some seldom-used character to be translated into a space when output is generated.

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 statement 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. ◆

Hyphenating text

Formatters do not perform hyphenation unless it is 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. "Controlling Format Style of Footnote Text" later in this chapter 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 two-character sequence \%) appears at the beginning of a word, the word is not hyphenated. Alternatively, this sequence can be used to indicate legal hyphenation points inside a word. All occurrences of the hyphenation indicator disappear when output is generated.

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 or phrase containing hyphens or dashes (also known as **em** dashes) will be hyphenated immediately after a hyphen or dash if it is necessary to hyphenate, 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

Setting tabs

Macros .MT (see "Understanding Memorandum Types" later in this chapter), .TC, and .CS (see "Generating a Table of Contents and Cover Sheet" later in this chapter) 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 1/2 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, . . . , and 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 (see "Understanding Formatting" earlier in this chapter).

The `tbl`(1) program (see "Using Displays in Tables" later in this chapter) changes tab stops but does not restore default tab settings.

Justifying the right margin

The `.SA` macro is used to set right-margin justification for the main body of text.

`.SA` [*arg*]

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:

0   Sets both flags to no justification, the same as 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.

Spacing lines of text

`.SP` [*lines*]

There are 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, in which case 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 use `.sp` for spacing. For example, `.LE 1` (see "Using List-Item Macros" later in this chapter) immediately followed by `.P` (see "Creating Paragraphs") 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.

Setting point size and vertical spacing

The prevailing point size and vertical spacing can be changed by invoking the `.s` macro:

`.s` [*point size*] [*vertical spacing*]

In the `troff` formatter, the default point size obtained from the mm register s is 10 points; the vertical spacing is 12 points, six lines per inch. The mnemonics D (default value), C (current value), and P (previous value) can be used for both arguments. See "Parameters Set From the Command Line" earlier in this chapter for an alternative way to set these parameters.

In the `troff` formatter, these guidelines apply:

- 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 D, the default, is used for the vertical spacing.

Default value for vertical spacing is always two points greater than the current point size. Footnotes are two points smaller than the body with an additional 3-point space between footnotes. A null (" ") value for either argument defaults to c, the 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
```

Reducing point size of a string

The .SM macro allows the user to reduce by one point the size of a string.

.SM *string1* [*string2*] [*string3*]

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 is null), the second argument is made smaller, and all three arguments are concatenated. For example,

```
.SM X
```

produces

```
X
.SM Y XYX ""
```

produces

```
YXYX
```

and

```
.SM ( YXYX )
```

produces

```
(YXYX)
```

Creating 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 uses this string to generate automatically the bullets for bullet-listed items (see "Creating a Bulleted List" later in this chapter).

Using 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 (that is, `\(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 hyphens (--) in the `nroff` formatter. Dash list (`.DL`) macros (see "Creating a Dashed List" later in this chapter) 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 (–).

Using bold, italic, and roman fonts

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.

`.B` [*bold-arg*] [*previous-font-arg*] . . .

`.I` [*italic-arg*] [*previous-font-arg*] . . .

`.R`

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 you are 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. ◆

Creating a 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
A/UX\*(Tm manual
is available from the library.
```

yields

The A/UX™ manual
is available from the library.

Producing accents

Strings can be used to produce accents for letters as shown in the following examples:

	Input	*Output*
Grave accent	e*`	è
Acute accent	e*	é
Circumflex	o*	ô
Tilde	n*	ñ
Cedilla	c*	ç
Lowercase umlaut	u*¨	ü
Uppercase umlaut	U*;	Ü

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 end-of-file character) also allows the user to resume normal output. See stty(1) in *A/UX Command Reference* for information about the user-specifiable sequences.

Using 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, those listed in Table 4-3.

Table 4-3 Formatter requests useful with mm

Request	Description
.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.

Structuring the page

Using mm macros you can create indented and numbered paragraphs, establish headings and change their appearance, create customized headers and footers, change the text flow to two-column output, and use a variety of other macros to create the layout that best suits your purposes.

Creating paragraphs

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

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

Indenting paragraphs

An indented or an unindented paragraph is defined with the type argument

0   Left justified
1   Indented

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 (see "Creating Numbered Headings" later in this chapter). 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 when they occur 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 0 for any paragraphs to be indented.

◆ **Note** Values that specify indentation must be unscaled and are treated as character positions, that is, 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 (see "Creating Lists" later in this chapter).

Numbering paragraphs

Numbered paragraphs may be produced by setting the `Np` register to 1. This produces paragraphs numbered within first-level headings, for example, 1.01, 1.02, 1.03, 2.01, and so forth.

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.

Setting 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.

Creating 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 quotation marks.

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 (see "Working With Unnumbered Headings" later in this chapter) because the .H macro also performs the function of the .P macro. Any .P macro immediately following a .H macro is ignored. It is, however, good practice to start every paragraph with a .P macro, thereby ensuring that all paragraphs begin with a .P throughout a document.

Using 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 an underlined (italicized) font heading, followed by a single blank line. 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` *n heading-text*	Produces an underlined (italicized) font heading followed by two spaces ($3 \le n \le 7$). The following text begins on the same line; that is, 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 "Working With Unnumbered Headings" later in this chpater. ◆

Changing the appearance of headings

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.

Prespacing headings and forcing a page break A first-level heading (`.H 1`) normally has two blank lines (one vertical space) preceding it, and all other headings are preceded by one blank line (`nroff`) or one-half a vertical space (`troff`). If a multiline heading is 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; that is, a page eject occurs if the heading level is less than or equal to the `Ej` value.

Setting 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` (postheading indent).

If the heading level is less than or equal to the value of Hb, a break (see "Understanding Formatting" earlier in this chapter) 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, that is, 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 (see "Indenting Paragraphs" earlier in this chapter).

- 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
```

Centering 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).

Emphasizing headings with bold, italics, and underlining 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. You can use any font number defined on your output device, for example:

Formatter	HF code 1	2	3	Default HF code
`nroff`	No underline	Underline	Bold	2 2 2 2 2 2 2
`troff`	Roman	Italic	Bold	2 2 2 2 2 2 2

Thus, levels 1 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
```

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

- The normal style (`.ul` request) is used 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 option when invoking the `nroff` formatter (see "Parameters Set From the Command Line" earlier in this chapter).

Setting point sizes for headings The user can 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 0 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 or .HZ) may cause overprinting. ◆

Marking styles: Numerals and concatenation 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.

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

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 described in Table 4-4.

Table 4-4 Arguments for marking numeral styles

Argument	Meaning
1	Arabic (default for all levels)
0001	Arabic with enough leading zeros 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, that is, 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
```

Working with unnumbered headings

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.

.HU *heading-text*

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 0 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 appendixes and other sections that may not fit well into the numbering scheme of the main body of a document (see "Sample Appendix Headings" later in this chapter).

Using headings in the table of contents

The text of headings and their corresponding page numbers can be collected automatically 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 (see "Generating a Table of Contents and Cover Sheet" later in this chapter) at the end of the document

Any heading whose level is less than or equal to the value of the c1 register is saved and later displayed in the table of contents. The default value for the c1 register is 2; that is, 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 and footnotes (see "Creating Displays" and "Creating Footnotes" later in this chapter). If this happens, the "Out of temp file space" formatter error message will be issued; the only remedy is to save fewer levels, to have fewer words in the heading text, or do both.

Using headings in page numbering

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 option on the command line (see "Using Default Headers and Footers With Section-Page Numbering" later in this chapter). 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.

Creating user exit macros

This section 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 specify these features

again. All default actions occur if these macros are not defined. If `.HX`, `.HY`, or `.HZ` is defined by the user, user-supplied definition is interpreted at the appropriate point. These macros can influence handling of all headings because the `.HU` macro is actually a special case of the `.H` macro.

If the user first invokes the `.H` macro, then the derived level argument (*dlevel*) and the real level argument (*rlevel*) both are equal to the level given in the `.H` invocation. If the user first invokes the `.HU` macro (see "Working With Unnumbered Headings" earlier in this chapter), *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 (see "Prespacing Headings and Forcing a Page Break" earlier in this chapter), and accumulated the "heading mark," that is, 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 (see "Setting Spacing After Headings" earlier in this chapter).
	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>
```

- Ensure 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 (see "Naming Conventions" later in this chapter).

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, for example,

```
.de HZ
.if \\$1=1 .PF "Section \\$3"
```

Creating page headers and footers

Text printed at the top of each page is called a **page header.** Text printed at the bottom of each page is called a **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.

Using default headers and footers

By default, each page has a centered page number as the header. There is no default footer and no even or odd default headers or footers except as specified in the next section, "Using Default Headers and Footers With Section-Page Numbering."

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 type (.MT) macro:

- memorandum and released-paper style document macros (.TL, .AU, .AT, .TM, .AS, .AE, .OK, .ND, .AF, .NS, and .NE)

- page header and footer macros (.PH, .EH, .OH, .PF, .EF, and .OF)

- the .nr and .ds requests

Using default headers and footers with section-page numbering Pages can be numbered sequentially within sections by section number and page number (see "Using Headings in Page Numbering" earlier in this chapter). 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, for example, 7-2—and the default page header is blank.

Using 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 an 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.

Page headers The `.PH` macro specifies the header that is to appear at the top of every page.

`.PH` [*arg*]

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 option `-rD1` on the command line, 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 followed by the current line number within the current input file). (See "Parameters Set From Command Line" and "SCCS Release Identification.")

Even-page headers The `.EH` macro supplies a line to be printed at the top of each even-numbered page immediately following the header.

`.EH` [*arg*]

Initial value is a blank line.

Odd-page header The `.OH` macro is the same as `.EH` except that it applies to odd-numbered pages.

`.OH` [*arg*]

Page footers The `.PF` macro specifies a line that is to appear at the bottom of each page.

`.PF` [*arg*]

Its initial value is a blank line. If the `-rc`*n* flag option is specified on the command line, 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 instead of being a blank line.

Even-page footers The `.EF` macro supplies a line to be printed at the bottom of each even-numbered page immediately preceding the footer.

`.EF` [*arg*]

Initial value is a blank line.

Odd-page footers The .OF macro supplies a line to be printed at the bottom of each odd-numbered page immediately preceding the footer.

.OF [*arg*]

Initial value is a blank line.

First-page footers By default, the first-page footer is a blank line. If, in the input text file, the user specifies .PF, .OF, or both, 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 option is specified on the command line (see "Parameters Set From the Command Line" earlier in this chapter).

Strings and registers in header and footer macros String and register names can 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:

.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

.PF "'''– \\\\n(H1–\\\\nP –'"

If the user arranges for the string a] to contain the current section heading that 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.

Header and footer example

The following sequence specifies blank lines for header and footer lines, page numbers on the outside margin of each page (that is, 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'"
```

Skipping pages

The .SK macro skips pages but retains the usual header and footer processing.

.SK [*pages*]

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 (in which case 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.

Forcing an odd page

The .OP macro is used to ensure that formatted output text following the macro begins at the top of an odd-numbered page.

.OP

- 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-numbered page, text resumes printing at the top of the next page.

- If currently on an odd-numbered 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.

Specifying top and bottom margins

The .VM (vertical margin) macro allows the user to specify additional space at the top and bottom of the page.

.VM [top] [bottom]

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).

Using the word "PRIVATE" in the header

.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*:

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.

Defining a macro for top-of-page processing

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 (see "Creating Headings for Two-Column Output" later in this chapter). 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, that is, document number, issue date, and revision date, the user could define the .TP macro as follows:

```
.de TP
.sp
.ce 3
777-888-999
Iss. 2, AUG 1977
Rev. 7, SEP 1977
.sp
..
```

The .PX macro can be used to provide text that is to appear at the top of each page after the normal header and that can have tab stops to align it with columns of text in the body of the document.

Defining a macro for bottom-of-page processing

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.

```
.BS
```
zero or more lines of text
```
.BE
```

This block of text is removed by specifying an empty block, that is,

```
.BS
.BE
```

The bottom block will appear on the table of contents, pages, and cover sheet for memorandum for file, but not on the technical memorandum or released-paper cover sheets.

Creating a disclaimer using a proprietary marking macro

The `.PM` (proprietary marking) macro appends a proprietary disclaimer to the page footer. The proprietary disclaimers are constructed from strings defined in the file `/usr/lib/macros/strings.mm`.

```
.PM [code]
```

The argument is selected from among the following:

```
PM1
PM2 or CA
PM3 or CP
PM4
PM5
PM6
```

Use `.PM` at the beginning of your document, before you use footnotes or macros that define the memorandum style. Otherwise, an interaction between this macro and another that redefines the appearance of the bottom of the page may cause you problems.

The default disclaimers are in a form approved for use by AT&T. Markings are underlined. (They are italicized in `troff`.)

System administrators can change the contents of the `string.mm` file to match your needs. This file is described in "Using Define File Information" later in this chapter. In cases where the disclaimer message for a code argument has been removed, the argument issues a currently approved disclaimer message. Because the code argument may produce a shorter or longer disclaimer message, the page formatting of the document may be affected.

Creating two-column output

The `.2c` macro begins two-column processing, which continues until a `.1c` macro (one-column processing) is encountered.

```
.2c
```
text and formatting requests (except another `.2c`)
```
.1c
```

In two-column processing, each physical page is thought of as containing two-columnar "pages" of equal (but smaller) "page" width. Page headers and footers are not affected by two-column processing. The `.2c` macro does not balance two-column output.

It is possible to have full-page-width footnotes and displays when in two-column mode, although default action is for footnotes and displays to be narrow in two-column mode and wide in one-column mode. Footnote and display width is controlled by the `.wc` (width control) macro, which takes the arguments listed in Table 4-5.

Table 4-5 Arguments for the width control macro

Argument	Meaning
N	Default mode (`-WF`, `-FF`, `-WD`, `FB`).
WF	Wide footnotes (even in two-column mode).
-WF	Default: Turn off `WF`. Footnotes follow column mode; wide in one-column mode (`1C`), narrow in two-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 two-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`. ◆

Creating headings for two-column output

This section is intended only for users accustomed to writing formatter macros.

In two-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 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^Icenter^Iright^Ileft^Icenter^Iright
^Ifirst column^I^I^Isecond column
.fi
.sp 2
..
```

where ^I 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. See "Defining a Macro for Top-of-Page Processing" earlier in this chapter for more information on headers.

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; that is, 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 option.

◆ **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. ◆

Creating lists

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`) identify unique items to the system. They are followed by the actual text of the corresponding list items.

- 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, and so forth); 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 (see "Using List-Begin Macros" and "Defining Other List Structures" later in this chapter).

Using list-initialization macros

List-initialization macros are implemented as calls to the more basic `.LB` macro (see "Using List-Begin Macros" later in this chapter). The list-initialization macros are listed in Table 4-6.

Table 4-6 List-initialization macros

Macro	Description
.AL	Automatically numbered or alphabetized list
.BL	Bulleted list
.DL	Dashed list
.ML	Marked list
.RL	Reference list
.VL	Variable-item list

Using list-item macros

The .LI macro is used with all lists and for each list item.

.LI [*mark*] [1]

one or more lines of text that make up the 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 (see "Specifying Unpaddable Spaces" earlier in this chapter).

If the current *mark* (in the current list) is a null string and the first argument of `.LI` is omitted or null, the result is that of a "hanging indent"; that is, the first line of the following text is moved to the left starting at the same place where *mark* would have started (see "Creating a Variable-Item List" later in this chapter). ◆

Using list-end macros

The `.LE` macro restores the state of the list to that existing just before the most recent list-initialization macro call.

```
.LE [1]
```

If the optional argument is given, the `.LE` generates 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 (for example, by insertion of text).

Setting spacing in a list

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 6 (which is the maximum list-nesting level).

Numbering or alphabetizing a list

The .AL macro is used to begin sequentially numbered or alphabetized lists.

.AL [*type*] [*text-indent*] [1]

   If there are no arguments, the list is numbered, and text is indented by Li (default is 6) 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, that is, 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 the *type* argument is omitted or null, the value 1 is assumed. Listed below are the arguments and interpretations:

Argument	*Interpretation*
1	Arabic (default for all levels)
A	Uppercase alphabetic
a	Lowercase alphabetic
I	Uppercase Roman
i	Lowercase Roman

B. If the *text-indent* argument is non-null, it is used as the number of spaces from the current indent to the text; that is, it is used instead of the Li register for this list only. If the *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. However, a blank line will occur before the first item.

Creating a bulleted list

The .BL macro begins a bulleted list.

.BL [*text-indent*] [1]

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 (see "Creating Paragraphs" earlier in this chapter). In the default case, the text of a bulleted 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.

Creating a dashed list

The .DL macro begins a dashed list.

.DL [*text-indent*] [1]

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 (see "Creating Paragraphs" earlier in this chapter). In the default case, the text of a dashed 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.

Creating a marked list

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.

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

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 (see "Specifying Unpaddable Spaces" earlier in this chapter). ◆

Creating a reference list

A .RL macro call begins an automatically numbered list in which the numbers are enclosed by square brackets ([ ]).

.RL [*text-indent*] [1]

The string .RL 5 is used to initialize the following list:

[1] If the *text-indent* argument is specified (non-null), it is used as the number of spaces from the current indent to the text; that is, it is used instead of Li for this list only. If the *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.

Creating a variable-item list

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.

`.VL` *text-indent* [*mark-indent*] [1]

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 .br 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, it 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 (see "Specifying Unpaddable Spaces" earlier in this chapter). If you do not escape the spaces within the double quotation marks containing the list item, the first line of the text will be slightly adjusted for the paddable spaces and will not line up with the rest of the text blocks in your list. ◆

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 (see "Numbering or Alphabetizing a List," and "Creating a Dashed List" earlier in this chapter) 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 dashed list item.
.LI + 1
This is another dashed item in the same list
as the above item with a "plus" as prefix.
This is the last item in the dashed 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
alphabetized 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 dashed list item.

    + — This is another dashed item in the same list as the above item with a "plus" as prefix.
          This is the last item in the dashed 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 alphabetized list.

Using list-begin macros

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).

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

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 bulleted and dashed 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 (that is, 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 the 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 or characters found in the *mark* argument. If *type* is greater than 0, automatic numbering or alphabetizing is done. Then, *mark* is interpreted as the first item in the sequence to be used for numbering or alphabetizing and is chosen from the set (1, A, a, I, i), as in "Numbering or Alphabetizing a List" earlier in this chapter. This is summarized in the following list:

Type	Argument mark	Result
0	Omitted	Hanging indent
0	*String*	*String* is the *mark*
>0	Omitted	Arabic numbering
>0	One of 1, A, a, I, or i	Automatic numbering or alphabetic sequencing

Each nonzero value of *type* from 1 to 6 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** *mark* must not contain ordinary (paddable) spaces because alignment of items will be lost if the right margin is justified (see "Specifying Unpaddable Spaces" earlier in this chapter). ◆

■ The *LI-space* argument gives the number of blank lines (nroff) or half vertical spaces (troff) that should be generated 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 generated 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.

The following section, "Defining Other List Structures," shows how to build upon the supplied list of macros to obtain other kinds of lists.

Defining other list structures

This section 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 macros, 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 g \\n(:g\}
    \" 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.

Creating memorandum and released-paper style documents

Some of the information in this section is applicable to Bell Laboratories documents only. However, most of the features discussed here can be tailored to specific needs.

One use of the memorandum macros is the preparation of memoranda and released-paper documents that have special requirements for the first page and for the cover sheet. Data needed (title, author, date, case numbers, and so forth) is entered in the same way for both styles; an argument to the .MT macro indicates which style is being used.

Understanding the sequence of beginning macros

If the following macros are present, they must be given in the following order:

.ND *new-date*

.TL [*charging-case*] [*filing-case*]
 one or more lines of title 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 the 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.

Generating a title

The .TL macro generates a centered 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 quotation marks.

- 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 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 "Filing Case" will appear after the title.

Describing the author

The .AU and .AT macros take arguments that describe an author.

.AU *name* [*initials*] [*loc*] [*dept*] [*ext*] [*room*] [*arg*] [*arg*]

.AT [*title*] . . .

If any argument contains blanks, that argument must be enclosed within double quotation marks. 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 (see "Creating End-of-memorandum macros" later in this chapter) 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" SJJ 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 can be suppressed on the first page of a memorandum by setting the Au register to 0; the default value for Au is 1. The seventh through ninth rguments 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 can be used for organizational numbering schemes, and so on, for example,

```
.AU "S. P. Lename" SPL IH 998 7766 5H-44 654-3210.01MF
```

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 in 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 contains information regarding authors from different locations (see "Understanding Memorandum Types" later in this chapter). ◆

Specifying the TM numbers

If the memorandum is a technical memorandum, the TM numbers are supplied via the .TM macro.

.TM [*number*] . . .

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 (see "Understanding Memorandum Types" later in this chapter).

Identifying the abstract

If a memorandum has an abstract, the input is identified with the .AS (abstract start) and .AE (abstract end) delimiters.

.AS [*arg*] [*indent*]
one or more lines of abstract text
.AE

Abstracts are printed on page one of a document, on its cover sheet, or on both. There are three styles of cover sheet:

- released paper
- technical memorandum
- memorandum for file (also used for engineer's note, memorandum for record, and so on)

Cover sheets for released papers and technical memoranda are obtained by invoking the .cs macro (see "Generating a Table of Contents and Cover Sheet" later in this chapter).

In released-paper style (first argument of the .MT macro is 4) and in technical memorandum style, if the first argument of .AS is

0   Abstract will be printed on page one and on the cover sheet (if any).

1   Abstract will appear only on the cover sheet (if any).

(See "Understanding Memorandum Types" later in this chapter.)

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 one, and no cover sheet will be printed.

2   Abstract will appear only on the cover sheet, which will be produced automatically (that is, 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 are allowed on memoranda for file cover sheets; the .NS and .NE macros must appear after the .AS 2 and .AE macros. Headings and displays are not permitted within an abstract. (See "Creating Numbered Headings" and "Working with Unnumbered Headings" earlier in this chapter and "Using Copy To and Other Notations" and "Creating Displays" later in this chapter.)

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," that is, as the number of ens.

Using other keywords

Topical keywords may be specified on a technical memorandum cover sheet using the .OK macro.

.OK [*keyword*] ...

Up to nine such keywords or keyword phrases can be specified as arguments to the .OK macro; if any keyword contains spaces, it must be enclosed within double quotation marks.

Understanding memorandum types

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.

`.MT` [*type*] [*addressee*]

The *type* arguments and corresponding values are

`" "`	No memorandum type printed
`0`	No memorandum type printed
None	`MEMORANDUM FOR FILE` is printed
`1`	`MEMORANDUM FOR FILE` is printed
`2`	`PROGRAMMER'S NOTES` is printed
`3`	`ENGINEER'S NOTES` is printed
`4`	Released-paper style
`5`	External-letter style
`"string"`	string is printed

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 cannot 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, 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 used, most Bell Telephone Laboratories 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 (see "Creating End-of-memorandum macros" later in this chapter) 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 (see "Using an Alternate First-Page Format" later in this chapter) and defining a string (with a two-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, 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 with the company logo and address of the author will be used.

Changing the date

The .ND macro alters the value of the string DT, which is initially set to produce the current date.

.ND *new-date*

    If the argument contains spaces, it must be enclosed within double quotation marks.

Using an alternate first-page format

An alternate first-page format can be specified with the .AF macro.

.AF [*company-name*]

    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, allowing output on preprinted stationery. The use of .AF with no arguments is equivalent to the use of -rA1 except that the latter must be used if it is necessary to change the line length, page offset, or both (these default to 5.8i and 1i, respectively, for preprinted forms). The flag options -rOk and -rWk are not effective with .AF. The only .AF use appropriate for the troff formatter is to specify a replacement for "Bell Laboratories." The flag option -rE*n* controls the font of the "Subject/Date/From" block. (See "Parameters Set From the Command Line" earlier in this chapter).

Example of input text

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 Rev.)" ECP PY
```

```
.AU "N. W. Smith (June 1980 Rev.)" NWS PY
.MT 4
```

Figures 4-1, 4-2, and 4-3 later in this chapter show the input text file for a simple letter as well as the formatted output from both the `nroff` and `troff` formatters.

Creating end-of-memorandum macros

At the end of a memorandum 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.

Using the signature block

The `.FC` and `.SG` macros print a formal closing and 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 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 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
```

Using "copy to" and other notations

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.

.NS [arg]
zero or more lines of the notation
.NE

Codes for *arg* and the corresponding notations are listed in Table 4-7.

Table 4-7 "Copy to" notations

Argument	Notation
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 one exists. 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 can also be used following .AS 2 and .AE to place the notation list on the memorandum for file cover sheet (see "Identifying the Abstract" earlier in this chapter). If notations are given at the beginning without .AS 2, they will be saved and generated at the end of the document.

Generating the approval signature line

The .AV macro can be used after the last notation block to automatically generate a line with spaces for the approval signature and date.

.AV *approver's-name*

    For example,

.AV "Jane Doe"

produces

APPROVED:

Jane Doe   Date

Forcing a one-page letter

To increase the page length temporarily, for example, to force space for a signature at the bottom of a letter, you can use the −rL*n* flag option. For example, using −rL90 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.

◆ **Note** This will work only for a single-page letter or memo. ◆

Using define file information

The /usr/lib/macros/strings.mm file contains predefined strings for the .MT and .PM macros. These strings are proprietary disclaimers for AT&T Bell Laboratories and may be redefined by system administrators to contain different string and font information. Only system administrators have write permissions to change the define file.

Using business letter style

An alternative to the format memorandum style is the business letter style, which produces four types of business letters: blocked, semiblocked, full-blocked, and simplified.

Using the letter-type macro

The letter-type macro .LT formats a letter in one of four business styles:

.LT [*arg*]

.LT accepts one optional argument. Arguments and corresponding formats are listed in Table 4-8.

Table 4-8 Letter-type arguments and formats

Argument	Format
None	Blocked
BL	Blocked
FB	Full-blocked
SB	Semiblocked
SP	Simplified

.LT controls the placement on the page of the output of the subordinate macro .LO and the subordinate macro pairs (.IA and .IE, .WA and .WE), which differs according to each of the four business letter formats.

Business letter and formal memorandum macros (.LT and .MT) are mutually exclusive. If you specify both .LT-specific and .MT-specific macros in a single document, nroff/troff attempts to process the file according to the first formatting specific macro it encounters. mm ignores .MT-specific macros and .MT-specific command-line parameters if you use them with .LT; conversely, if you use .LT-specific macros with .MT, mm ignores them.

If you use these business letter macros, the macro pairs .WA/.WE, and .IA/.IE and the page formatting macros .LT are required; all other business letter macros are optional.

The .LT macro arguments control paragraph indentation for each of the four letter types. If you redefine the Pt and Pi registers, the user-specified indentations will override. Specification of the Pt and Pi registers must occur after specification of the .LT macros.

- In the block format all lines of text begin at the left margin except the dateline, return address, closing, and writer's identification. These begin at the center of the line. (The center of the line is not a fixed point; it is calculated for the current line length.)

- The semiblocked format is the same as the blocked format, except the first line of each paragraph is indented five spaces.

- In full-blocked format all lines begin at the left margin. There are no exceptions.

- The simplified format is the same as the full-blocked format, except the salutation is replaced by an all-capital subject line and is followed by an additional blank line, the closing is omitted, and the writer's identification is in capital letters on one line.

Table 4-9 presents a synopsis of the placement of business letter components for the four .LT letter formats and lists the macros (which are explained in detail below) that you use to format those components.

There are two possible error conditions for the .LT macro:

- If you omit the .LT macro, file processing aborts and an appropriate error message print

- If mm does not recognize an argument to .LT, the file processing aborts and an appropriate error message prints.

Table 4-9 Letter formatting components and macros

Macro	Function	BL	SB	FB	SP
.WA/.WE	Writer's address	Center	Center	Left	Left
.LO CN [arg]	Confidential notation	Left	Left	Left	Left
.LO RN[arg]	Reference notation	Center	Center	Left	Left
.IA/.IE	Inside address	Left	Left	Left	Left
.LO AT [arg]	Attention	Left	Left	Left	Left
.LO SA [arg]	Salutation	Left	Left	Left	None
.LO SJ [arg]	Subject line	Left	Indented	Left	Left
.P	Paragraphs	Left	Indented	Left	Left
.FC	Closing	Center	Center	Left	Left
.SG	Signature	Center	Center	Left	Left
.NS/.NE	Copy notation	Left	Left	Left	Left

Using writer's address macros

Use this macro pair to specify the writer of the letter and the writer's return address.

.WA *writer-name* [*title*]
return address
.WE

For example,

```
.WA "James Lorrin, Ph.D." Director
Summit Research Company
38 River Road
Summit, New Jersey 07901
.WE
```

If a complete return address is not necessary for the letter (for example, if you use printed letterhead stationary), you can specify the writer information alone:

```
.WA "James Lorrin, Ph.D." Director
.WE
```

The return address cannot exceed 14 lines. Lines in the return address that follow line 14 do not appear on the letter.

The two arguments specified for the .WA and .WE macro pair, the *writer-name* and the *title*, provide information used by the .SG macro. If you do not specify the .SG macro, the writer's name does not appear on the letter.

For the case of multiple writers on a single letter, you may specify only one writer return address. The specified writer return address must appear with the first *writer-name* as the first invocation of the .WA/.WE macro pair. Later return address specifications do not appear on the letter, although any number of additional writer names may be specified, for example,

```
.WA "James Lorrin, Ph.D." Director
Summit Research Company
38 River Road
Summit, New Jersey 07901
.WE
.WA "John Smith" Supervisor
.WE
.WA "Diane Kane" "Technical Support"
.WE
```

For blocked and semiblocked letter styles, the writer return address begins on line 12 of the first page and each line begins at the center of the line. For the full-blocked and simplified letter styles, the writer return address begins on line 12 of the page and each line begins at the left margin.

◆ **Note** Top-of-page processing can be controlled directly through `nroff`. The beginning of the printed page is user-defined. See the requests `.wh` and `.ch` in Chapter 3, "`nroff/troff` Formatters." ◆

If you omit either or both of the `.WA` and `.WE` macros, the file processing aborts and an appropriate error message prints.

Using inside address macros

`.IA` and `.IE` are a macro pair that you use to specify the addressee and the addressee's address. There are two ways that you can use this macro pair:

```
.IA
```
text
```
.IE
```
or
```
.IA [addressee-name] [title]
```
text
```
.IE
```

For example,

```
.IA Fred Smith, Ph.D.
Columbia University
116th Street
New York, New York 10019
.IE
```

or

```
.IA "Fred Smith, Ph.D."
.IE
```

For all four styles of .LT, the inside address prints on the fifth line below the date (if a reference notation or confidential notation appears after the date, the inside address prints three lines below the notation), and each line begins at the left margin.

If you omit either or both of the .IA and .IE macros, the file processing aborts and an appropriate error message prints.

Using the letter-options macro

The letter-options macro provides the capability for specifying five common business letter components:

.LO *type* [*arg*]

The .LO macro takes care of placement and spacing of these letter components for each .LT letter format. .LO requires one argument to specify a letter component type and accepts one optional string argument to refine its action. .LO's arguments and their corresponding components are

AT	Attention
CN	Confidential notation
RN	Reference notation
SA	Salutation
SJ	Subject line

Confidential notation The confidential notation shows that a business letter should be read only by the person to whom it is addressed. The confidential notation appears on the second line below the date line of the letter and begins at the left margin for all letter formats.

If the optional string argument is present the specified string replaces the default, for example,

.LO CN "RESTRICTED"

The default of CN prints CONFIDENTIAL.

Reference notation The reference notation supplies specific information to be used by the addressee, for example,

.LO RN "meeting of 1/25"

The reference note appears two lines below the dateline of the letter or on the second line below any notation that follows the date and is left aligned with the dateline for all four letter formats.

RN provides a common format for including a reference note by printing the string "In reference to:" preceding the optional string argument to .LO. The format string "In reference to:" cannot be redefined. There is no default value for the optional argument.

Attention line The attention line directs the letter to the attention of a specific person or department, for example,

.LO AT "Dr. Smith"

The attention information appears on the second line below the inside address of the letter and begins at the left margin.

AT provides a common format for directing a letter to the attention of a specific person by printing the string "ATTENTION:" preceding the optional string argument to .LO. The format string "ATTENTION:" cannot be redefined. There is no default value for the optional argument.

Salutation The salutation specifies the letter's opening greeting. For the blocked, semiblocked, and full-blocked formats the salutation appears on the second line below the inside address (or on the second line below the attention line, if used). In the simplified letter format, the salutation is ignored.

The default of SA prints "To Whom It May Concern:" for the salutation. If the optional string argument is present, the specified string will replace the default, for example,

.LO SA "Dear Dr. Smith"

Subject line The subject line shows what the letter is about. In the blocked and full-blocked letter formats, the subject line information appears on the second line below the salutation and begins at the left margin. For the semiblocked format the subject line appears on the second line below the salutation and is indented five spaces. In the simplified letter format the subject line information appears in place of the salutation three lines below the inside address of the attention line; the salutation, if you use it, is ignored.

For the blocked, semiblocked, and full-blocked formats, SJ provides a common format for indicating what the letter is about by printing the string "SUBJECT:" preceding the optional string argument to .LO.

```
.LO SJ "Staff Meeting"
```

The format string "SUBJECT:" cannot be redefined. There is no default value for the optional argument.

For the simplified letter, the subject line string argument prints on the third line below the inside address or the attention line (a salutation is ignored if used).

If you specify the .LO macro without an argument or the argument you specify is unrecognized, the file processing aborts and an appropriate error message prints.

Generating multipage letters

The .LT macro controls the format for the first page of the letter. The letter macros will not alter the default nroff/troff page processing following the first page of the letter.

Understanding the sequence of beginning letter macros

Macros .WA, .WE, .IA, .IE, and .LT must be given in the order listed below. .LO can be specified multiple times with different argument *types*. The .LO argument *types* do not have to be in any specific order. All .LO requests must be specified before .LT.

.ND *new date*
.WA *writer's name* |*title*|
Return address
Street City, State Zip Code
Text
.WE
.IA
Addressee name
Title
Company
Street City, State Zip Code
Text
.IE

.LO *type*[*arg*]

.LT [*arg*]

.P

Text

.FC

.SG [*arg* [1]]

.NS [*arg* [1]]

Text

.NE

If you put `nroff`/`troff` requests and lines of text before `.LT`, you change how `.LT` works. For example, if the first line of a file is a line of text, mm processes the file as if you had not specified `.LT`.

Creating 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, "`nroff`/`troff` Formatters"). 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 can never be nested in any combination. Although lists and paragraphs are permitted, no headings (.H or .HU) can occur within displays or footnotes. ◆

Starting static displays

A static display is started by the .DS macro and terminated by the .DE macro.

.DS [*format*] [*fill*] [*rindent*]
one or more lines of text
.DE

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 meanings shown in Table 4-10.

- The *fill* argument is an integer or letter and can have the meanings shown in Table 4-11.

- The *rindent* argument is the number of characters that the line length should be decreased, that is, 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 it defaults to ems.

Table 4-10 Format argument in static displays

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

Table 4-11 Fill argument in static displays

Fill	Meaning
" "	No-fill mode
Omitted	No-fill mode
0 or N	No-fill mode
1 or F	Fill mode

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 (see "Creating Paragraphs" earlier in this chapter). Even though their initial values are the same default values, these two registers are independent.

The display *format* argument value 3 (or CB) horizontally centers 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 the 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 (see "Using Displays in Equations" later in this chapter).

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 (that is, 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."

Starting floating displays

A floating display is started by the `.DF` macro and terminated by the `.DE` macro.

`.DF` [*format*] [*fill*] [*rindent*].
one or more lines of text
`.DE`

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 Tables 4-12 and 4-13 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 generated. 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 generated. When a new page is started, or
the top of the second column in two-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 generated, 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 are generated. 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 two-column document.

Table 4-12 `De` number register code settings in floating displays

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.

Table 4-13 `Df` number register code settings in floating displays

Code	Action
0	Floating displays will not be generated until end of section (when section-page numbering) or end of document.
	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 generated (this also applies to code 5).
1	Generate new floating display on current page if there is space; otherwise, hold it until end of section or document.
2	Generate exactly one floating display from queue to the top of a new page or column (when in two-column mode).
3	Generate one floating display on current page if there is space; otherwise, output to the top of a new page or column.
4	Generate as many displays as will fit (but at least one) starting at the top of a new page or column.
5	Generate a new floating display on the current page if there is room (default condition). Generate 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 generated.

The .WC macro (see "Creating Two-Column Output" earlier in this chapter) can also be used to control handling of displays in double-column mode and to control the break in text before floating displays.

Using displays in tables

The mm macros interact with the tbl macros and provide some extra functionality (see Chapter 7, "tbl Tables," for a description of the tbl program).

.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, the 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
```
causes 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 contains the table. This feature is used when a single table must be broken into segments because of table complexity (for example, 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 contains the table.

For the nroff formatter, the –e flag option (–E for mm(1)) can be used for terminals, for instance, 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 flag option is not effective with col(1). (See "The mm Command" earlier in this chapter.)

Using displays in equations

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, when processed with the mm macros, .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 inline 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.

```
.DS
.EQ [label]
equation(s)
.EN
.DE
```

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 can 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.

Using displays in figure, table, equation, and exhibit titles

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 print captions for figures, tables, and equations.

```
.FG [title] [override] [flag]
.TB [title] [override] [flag]
.EC [title] [override] [flag]
.EX [title] [override] [flag]
```

These macros use registers `Fg`, `Tb`, `Ec`, and `Ex`, respectively. (See "Parameters Set From the Command Line" earlier in this chapter 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 can 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` is given, "section-figure" numbering is set automatically, and the user-specified *override* argument is ignored. (See "Parameters Set From the Command Line" earlier in this chapter.)

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.

Listing figures, tables, equations, and exhibits

Lists 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 can 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
```

Creating 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, "nroff/troff Formatters").

Numbering footnotes

Footnotes may be automatically numbered by typing the three characters *F (that is, 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.

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 can 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 is not permitted between .FS and .FE macros. If footnotes are required in the title, abstract, or table (see "Using Displays in Tables" earlier in this chapter), 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 (see "Specifying Unpaddable Spaces" earlier in this chapter) to `\*F` or *label* when they follow an end-of-sentence punctuation mark (a period, question mark, or exclamation point).

Controlling format style of footnote text

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.

`.FD [`*arg*`] [1]`

The first argument (*arg*) is a number from the left column of Table 4-14. 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, "`nroff/troff` Formatters."

Table 4-14 Hyphenating footnote text

Argument	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 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.

Setting 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`) or a 6-point vertical space (`troff`) to occur between footnotes.

Generating a 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 (see "Identifying the Abstract" earlier in this chapter). ◆

These macros normally appear once at the end of the document, after the signature block and notations macros, and may occur in either order. (See "Using the Signature Block" and "Using Copy to and Other Notations" earlier in this chapter.)

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.

Generating a table of contents

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 `C1` register (see "Using Headings in the Table of Contents" earlier in this chapter).

`.TC` [*slevel*] [*spacing*] [*tlevel*] [*tab*] [*head1*] [*head2*] [*head3*] [*head4*] [*head5*]

Arguments to `.TC` control spacing before each entry, placement of associated page numbers, 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 `C1` register.

The third and fourth arguments (*tlevel* and *tab*) control placement of the 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 (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 (that is, 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:

<div align="center">

Special Application

Message Transmission

</div>

Approved:_____

<div align="center">

CONTENTS

.

.

.

</div>

If the `.TX` macro is defined as `.TY`, the word "CONTENTS" is 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 can be changed by defining the c_i 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 c_l register. Arguments must be scaled. For example, with c_l = 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, exhibits, and equations will be produced as separate pages unless Cp is set to 1, which causes these lists to appear on the same page as the table of contents.

Generating a cover sheet

The .CS macro generates a cover sheet in either the released-paper or technical memorandum style (see "Identifying the Abstract" earlier in this chapter for details of the memorandum for file cover sheet).

.CS [*pages*] [*other*] [*total*] [*figs*] [*tbls*] [*refs*]

All other information for the cover sheet is obtained from data given before the .MT macro call (see "Understanding the Sequence of Beginning Letter Macros" earlier in this chapter). If the technical memorandum style is used, the .CS macro generates the "Cover Sheet for Technical Memorandum." The data that appears in the lower left corner of the technical memorandum cover sheet (counts of pages of text, other pages, total pages, figures, tables, and references) is generated automatically (0 is used for other pages). These values can be changed by supplying the corresponding arguments to the .CS macro. If the released-paper style is used, all arguments to .CS are ignored.

Using 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.

Numbering references

Automatically numbered references can 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.

Delimiting reference text

The .RS and .RF macros are used to delimit text of each reference.

.RS [*string-name*]
.RF

    For example,

```
A line of text to be referenced.\*(Rf
.RS
reference text
.RF
```

Creating 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 that 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.

Generating a reference page

The `.RP` macro causes a reference page, entitled by default "References," to be generated automatically at the end of the document (before table of contents and cover sheet) and to be listed in the table of contents.

`.RP` [*arg1*] [*arg2*]

This page contains the reference items 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 (that is, 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 and 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"
```

Troubleshooting

This section explains what happens when a macro finds an error. This section also helps you find output that doesn't appear.

What happens when a macro detects an error?

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 "Error Messages" later in this chapter.
- Processing terminates unless register D has a positive value. In the latter case, processing continues even though the output is guaranteed to be deranged from that point on. (See "Parameters Set From the Command Line" earlier in this chapter.)

The error message is printed by generating the message directly to the user terminal. If an output filter, such as 300(1), 450(1), or hp(1), is being used to postprocess the nroff formatter output, the message may be garbled by being mixed with text held in that filter's output buffer.

◆ **Note** If any 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. ◆

Why does output disappear?

Disappearance of output usually occurs because of an unclosed diversion (for example, 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 backward from the termination point looking for the corresponding associated .DF, .DS, or .FS (because 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 is preceded by its filename and the line number in that file. This listing can be used to check for illegal nesting, omission of these macros, or both.

Extending and modifying memorandum macros

The naming conventions listed in this section allow you to extend and modify memorandum macros. 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.

Naming conventions

In this part, the following conventions are used to describe names:

a	Lowercase letter
A	Uppercase letter
n	Digit
s	Any nonalphanumeric character (special character)
x	Any alphanumeric character (*n*, *a*, or *A*, that is, letter or digit)

All other characters are literals; that is, they are characters that stand for themselves.

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)

Names used by memorandum macros

Macros and strings: *A, AA, Aa* (accessible to users, for example, 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; currently only the seven accents (see "Reducing Point Size of a String" earlier in this chapter)

                                    `)`*x*, `}`*x*, `]`*x*, `>`*x*, `?`*x* (internal)

Registers: *An, Aa* (accessible to users, for example, `H1`, `Fg`)

                                    A (accessible to users; meant to be set on the command line, for example, `c`)

                                    `:`*x*, `;`*x*, `#`*x*, `?`*x*, `!`*x* (internal)

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`. The 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*

Names defined by user

Names that consist of either 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* (for example, `bG`, `kW`)

Strings: *as* (for example, `c)`, `f]`, `p}`)

Registers: *a* (for example, `f`, `t`)

Sample 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 can be saved for the table of contents. To center appendix titles, the `Hc` register must be set to 1 (see "Centering Headings" earlier in this chapter).

5 ms Macros
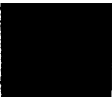
This chapter is a reference for the ms macro package. It's a good idea to skim this chapter for a general understanding of the ms macro package and then read specific sections in detail as needed.

What are ms macros?

ms is a collection of text-formatting macros for the A/UX text formatters nroff and troff. ms was designed for writing general-purpose documents. ms and me perform many of the same functions, but some features of me are not available in ms, so A/UX Release 3.0 supports both packages. You can use only one of these packages at a time, however, so you may wish to read this chapter and the chapter on me and make a decision about which package to use before you actually begin formatting a document.

For a complete discussion of text-formatting concepts and principles, refer to Chapter 1, "Introduction to A/UX Text Processing."

How input is read

Formatters fill output lines from one or more input lines. You can justify output lines so that both the left and right margins are aligned. As lines are being filled, words may also be hyphenated as necessary. You can turn any of these modes on and off (with the .na, .ad, .hy, .nf, and .fi formatter requests; turning off fill mode also turns off justification and hyphenation). Certain formatting commands (requests and macros) stop filling the current output line, print the line (of whatever length), and begin subsequent text on a new output line. This printing of a partially filled output line is called a break. A few formatter requests and most of the ms macros cause a break. (See Table 5-1.)

Table 5-1 ms macros that cause a break

Name	Description
.AB	Begin abstract.
.AI	Author's institution.
.AU	Author's name.
.BD	Block display (no keep).
.B1	Begin boxed text.
.B2	End boxed text.
.CD	Centered display (no keep).
.CT	Chapter title.
.DE	End display.
.DS	Start standard display.
.EN	End equation.
.EQ	Start equation.
.ID	Indented display (no keep).
.IP	Indented paragraph.
.KE	End keep.
.KS	Start keep.
.LD	Left-adjusted display (no keep).
.LP	Left-block paragraph.
.MC	Begin multcolumn text.
.NH	Numbered heading.
.PP	Standard paragraph.
.QP	Quotation mark paragraph.
.RE	End right shift.
.RS	Begin right shift.
.SH	Unnumbered section heading.
.TC	Print table of contents.
.TE	End table.
.TL	Print centered title in boldface.
.TS	Start table.
.XA	Additional index entry.
.XS	Begin index entry.
.1C	Resume one-column printing.
.2C	Begin two-column printing.

Understanding arguments and double quotation marks

In `ms`, you can use an argument to modify a macro. For example, the macro `.DS` begins a standard display. When you add a `c` to the macro

`.DS C`

the material in the display is centered.

Any macro argument containing ordinary (paddable) spaces must be enclosed in double quotation marks. A double quotation mark is a single character that must not be confused with two apostrophes, acute accents, or grave accents. If an argument containing such spaces is not enclosed in double quotation marks, it will be treated as several separate arguments.

Sequence of beginning macros

Any text file processed by the `ms` macros must begin with one of the following macros: `.TL`, `.SH`, `.NH`, `.PP`, and `.LP`.

These macros initialize the file and must precede a break caused by blank lines, leading spaces, or `.sp`, `.br`, and `.ce` `troff` requests.

Using basic document formats

The `ms` macro packages has facilities for formatting the basic elements of a document, such as the cover page, margins, and spacing.

Cover sheets

You can generate a separate cover sheet containing any of the following: title (`.TL`), author (`.AU`), author's institution (`.AI`), and abstract (`.AB`). Precede these macros with `.RP` and enter them in the order indicated. The current date is printed on the cover sheet (unless you suppress this feature with the `.ND` macro; see "Changing and Removing the Date" later in this chapter).

You can also include this information without producing a cover sheet. Title, author, abstract, and so on are then printed on the first page of the document.

Titles

The title macro (`.TL`) creates a centered title (as opposed to the three-part title format of the `troff` request `.tl`). In `troff` the title is printed two points larger than the remaining text and is in boldface. In `nroff` the title is underlined. When used with the `.RP` macro, the title is centered on the cover sheet. (See Table 5-2.)

Table 5-2 Title macro

Type	Form	Explanation
Macro	`.TL`	Print centered title in boldface two points larger than current font.

Authors

The macros `.AU` and `.AI` print the author's name and institution centered and in italic. (See Table 5-3.) For example,

`.AU`

author's-name

`.AI`

author's-institution

produces

<div align="center">

author's-name

author's-institution

</div>

Multiple authors (and institutions) can also be used. Precede each additional entry with .AU or .AI, as appropriate, for example,

.AU

author1

.AU

author2

Table 5-3 Author macros

Type	Form	Explanation
Macro	.AI	Print centered information about the author's institution.
Macro	.AU	Print centered author's name, in current point size and in italic. Multiple names are printed on separate lines if entered on separate input lines.

Abstracts

An abstract is a brief summary of the text it precedes. The .AB macro prints this summary after the author's institution, if used, with an optional centered heading. (See Table 5-4.)

Table 5-4 Abstract macros

Type	Form	Explanation
Macro	.AB [no]	Begin abstract. The abstract text is preceded by a centered heading titled "ABSTRACT." Argument no suppresses the heading. The abstract text is filled and adjusted on a line 5/6 the normal text line length.
Macro	.AE	End abstract.

Paper styles

You can produce cover sheets in two basic formats: standard released-paper or thesis mode. (See Table 5-5.)

Released paper format (.RP) provides a separate cover sheet containing title, author, institution, and abstract. (See "Cover Sheets" earlier in this chapter.)

Thesis mode (.TM) formats your document according to university specifications for doctoral dissertations. The page number is printed on each page, text is double-spaced, the current date is removed from the center footer, and the chapter title macro (.CT) is defined and activated.

Table 5-5 Paper styles macros

Type	Form	Explanation
Macro	.RP [no]	Released-paper format. Provides a separate cover sheet for title, author, author's institution, and abstract. This information is repeated on the first page of the document unless the argument no is specified.
Macro	.TM	Thesis mode. Automatically numbers each page; double-spaces all text except displays, quotation marks, and keeps; suppresses the printing of the date in the center footer; and defines the chapter title macro (.CT).

Chapter titles

The chapter title macro is defined only when you have invoked thesis mode. It begins a new page, moves the page number from the right header to the center footer, centers the lines that follow until a paragraph macro is reached, and, in the case of troff, prints these lines in boldface. (See Table 5-6.)

Table 5-6 Chapter title macro

Type	Form	Explanation
Macro	.CT	Move the page number from right header to the center footer, generate a page break, and center and boldface the lines following the request (thesis mode only).

UNIX trademark

You can insert the UNIX trademark in the text or in a footnote. (See Table 5-7.)

Table 5-7 UNIX trademark macro

Type	Form	Explanation
Macro	.UX	Prints "UNIX†" in the text plus a footnote that reads "UNIX is a trademark of UNIX System Laboratories, Inc."

Changing the look of the document

A document formatted with the ms macros is produced in a standard page layout. By default, text is generated in a single column, and a line of text is 6 inches from margin to margin. The left margin is 1 inch (in troff) from the edge of the paper, point size is set to 10 points, vertical space is set to 12 points, and tab stops are set every 5 spaces. The following macros and number registers permit you to change these default features and customize your page layout. You can also change fonts and remove the date.

Creating multicolumn output

Output from `troff` is normally a single column of text. Placing the `ms` command .2C in your file causes the output to be printed in two-column format. Each column is printed with a width of 7/15 of the current line length, and the gap between the two columns is 1/15 of the full line length.

To print text in more than two columns, use the .MC macro:

MC column-width gutter-width

The number of columns is computed automatically, based on the maximum number of columns of the specified width that can fit within the current line length. The *column-width* argument must be numeric, and unless indicated otherwise, the unit of measurement is assumed to be in ens.

The *gutter-width* argument permits you to control the distance between columns.

To return to single-column output, use the `.1C` command.

Any change in the number of columns specified (except from one to two or greater) causes a page break. (See Table 5-8.)

Table 5-8 Multicolumn macros

Type	Form	Explanation
Macro	`.2C`	Print text in two equal columns.
Macro	`.MC` *x y*	Print text in multiple columns. *x* is the column width, and *y* is the gutter width.
Macro	`.1C`	Restore one-column printing.

Setting point size and vertical spacing

Number registers are used to set default point size and vertical spacing. In `ms` the registers are called `PS` and `VS`. (To change relative point size using macros, see "Changing the String Point Size" later in this chapter). The defaults for point size and vertical spacing in the `ms` macro package are 10 and 12 points, respectively. The two-point difference allows for adequate spacing between lines.

When using ms, remember to change the vertical spacing register when changing the point size. Otherwise, the output will be either too widely or too closely spaced. (See Table 5-9.)

Table 5-9 Point size and vertical spacing registers

Type	Form	Explanation
Register	.PS	Point size Initial value: 10
Register	.VS	Vertical spacing Initial value: 12v

Changing top and bottom margins

By default, the distance between the header and footer text and the top and bottom edges of the paper is one inch. In ms, you can change these values by resetting registers HM and FM. (See Table 5-10.)

Table 5-10 Top and bottom margin registers

Type	Form	Explanation
Register	.HM	Vertical distance of the header margin Initial value: 1 inch
Register	.FM	Vertical distance of the footer margin Initial value: 1 inch

Changing line length

The default length of a line of text is six inches from left to right margin. In ms, you can change this by resetting the number register LL. (See Table 5-11.)

Table 5-11 Line length register

Type	Form	Explanation
Register	.LL	Line length Initial value: 6 inches

Changing page offset

The position of the left margin is determined by two dimensions: page offset and indentation. Indentation controls the current left margin, whereas page offset controls the absolute left margin.

Page offset is the distance betweeen the left margin and the left edge of the paper. Indentation is expressed as a distance to the right of page offset. You can change indentation within your document (see "Indenting Paragraphs" later in this chapter), but page offset is defined at the beginning of your document and usually remains constant throughout.

The default page offset is 1 in troff and 0 in nroff. In ms, you can change this by resetting number register PO. The value of number register PO multiplied by 2 plus the line length (register LL) must always equal 8, for example,

$1 \times 2 + 6 = 8$

where 1 is the default page offset and 6 is the default line length in troff. (See Table 5-12.)

Table 5-12 Page offset register

Type	Form	Explanation
Register	.PO	Absolute limit of the left margin Initial value: 1 inch in troff, 0 in nroff

Changing tab setting

In ms, you can set tabs with the .TA command. The default settings are in increments of 5 ens, but you may substitute any value needed. (See Table 5-13.)

Table 5-13 Tab setting macro

Type	Form	Explanation
Macro	.TA x	Set tabs to x, where x is the number of ens.
		Initial settings: increments of 5 ens

Changing fonts

You can use the following macros to emphasize words or groups of words. (See Table 5-14.) Typewritten or line-printed material is usually emphasized with underlining. Typeset and typeset-quality material is emphasized with **boldface** or *italics*.

In ms, the .B and .I macros produce boldface and italic, respectively, with troff and underlining with nroff.

.B or .I can be followed by RETURN, and all subsequent text will be printed in boldface or italic. This usage must be terminated by a .R command, indicating that printing should return to roman, as follows:

```
.B
This text will be printed in boldface.
.R
```

.B or .I can be followed by a single word on the same line.

The macros for boldfacing and italicizing can be followed by a group of words on the same line. These must be enclosed in double quotation marks. Again, only those words are emphasized, and no .R is needed. For example, you could use

.B "*group-of-words*"

The underline macros, .UL (ms) apply only to text processed with troff. They underline one word at a time. If multiple word underlining is desired, you must enter individual underlining commands for each word. Enclosing multiple words in quotation marks does not work. For example, in ms you could use

text

.UL *word1*

.UL *word2*

.UL *word3*

Table 5-14 Font changing macros

Type	Form	Explanation
Macro	.B [x]	Print x in boldface (troff only). If x is not present, print all subsequent text in boldface.
Macro	.I [x]	Print x in italic. If x is not present, print all subsequent text in italic.
Macro	.R	Return to roman font.
Macro	.UL x	Underline x (troff only).

Changing the string point size

In ms, three macros are provided to control the relative size of troff type. (See Table 5-15.) .SM and .LG decrease and increase point size by two points, respectively, and both can be repeated to increase the effect. The .NL command restores point size to the default. These macros are used for temporary size changes for a single word or a small group of words. (See "Setting Point Size and Vertical Spacing" earlier in this chapter to change absolute point size.)

Table 5-15   String point size changing macros

Type	Form	Explanation
Macro	.LG	Increase point size by 2.
Macro	.NL	Set point size back to normal.
		Initial value: 10
Macro	.SM	Decrease point size by 2.

Changing and removing the date

When you use nroff with the ms macros, the current date is printed at the bottom center of every page. With both nroff and troff, when you use .RP format (see "ms Paper Styles" earlier in this chapter), the current date is printed on the cover sheet of the document. The following macros are provided to change these default features. Use the .ND macro to suppress printing of the date. If you add a date as an argument, that date is printed on the cover sheet in released paper format. (See Tables 5-16 and 5-17.)

Table 5-16 Date changing macro

Type	Form	Explanation
Macro	.DA [x]	In troff print the current date at the bottom of each page (in nroff this is the default). Argument x replaces the current date with a different value. The current date is kept in string register *(DY.

Table 5-17 Date removal macro

Type	Form	Explanation
Macro	.ND [x]	Suppress printing of the date. If a date is given as an argument, it is printed on the cover sheet in .RP format.

Structuring the page

Using ms macros, you can create indented and labeled paragraphs, establish headings and change their appearance, create customized headers and footers, and control page breaks to create the layout that best suits your purposes.

Creating paragraphs

The ms macro package provides several commands that determine the style of your paragraph. In all cases, the formatter skips one vertical space before generating the text of the paragraph.

Creating the standard paragraph

The first line of a standard paragraph is indented. All other lines are generated at the left margin. The default indentation is 5 ens, but can be changed by setting the number register P I (see "Indenting Paragraphs" later in this chapter and Table 5-18).

Table 5-18 Standard paragraph macro

Type	Form	Explanation
Macro	. PP	Standard paragraph. The first line is indented the value of register P I (5 ens). The paragraph is preceded and followed by a vertical space equal to the value set in register P D (1v).

Creating a left-block paragraph

The text of a left-block paragraph is generated as a left-adjusted block. (See Table 5-19.)

Table 5-19 Left-block paragraph macro

Type	Form	Explanation
Macro	.LP	Left-block paragraph. The paragraph is offset vertically by the value of register PD (1v).

Indenting paragraphs

All lines of an indented paragraph are indented a certain value. (See Table 5-20.) In ms, the .IP command can be used in three ways:

.IP

.IP label

.IP label value

The first example produces a basic indented paragraph. Text is generated as a block five spaces from the left margin.

The other two forms of the indented paragraph command permit you to label your paragraph with some alphanumeric character. These can be used to produce numbered or bulleted lists. For example,

```
.IP (1)
This is a labeled indented paragraph.
```

produces

(1) This is a labeled indented paragraph.

You can substitute any character for the number. For example,

```
.IP *
This is a labeled indented paragraph.
```

produces

* This is a labeled indented paragraph.

You can also assign a value for the indentation level:

```
.IP (1) 10
```

Instead of the default indentation (5 ens), the formatter now indents the text 10 ens.

Table 5-20 Indented paragraph macro

Type	Form	Explanation
Macro	.IP [xy]	Indented paragraph, where *x* is the label, and *y* is the indentation. Default indentation is 5 ens, as set in the register P I. The paragraph is offset vertically by the value of register PD (1v).

In ms, the registers P I and Q I determine the amount of indentation for paragraphs. Values for each must be unscaled and are always read as ens. For example,

nr PI 1i

will not work. The value 1i (1 inch) will not be understood by troff; it must be given in ens.

   P I sets the indentation level for all paragraphs except quotation mark paragraphs. For quotation mark paragraphs, use the Q I form. (See Table 5-21.)

Table 5-21 Indented paragraph registers

Type	Form	Explanation
Register	.PI	Paragraph indentation. The values must be unscaled and are read as ens. Initial value: 5 ens
Register	.QI	Quotation mark paragraph indent. The values must be unscaled and are read as ens. Initial value: 5 ens

Creating a hanging paragraph

The first line of text in an exdented paragraph (hanging indent) is flush with the left margin; all subsequent lines are indented the default 5 ens. This ms macro is often used to format bibliographic references. (See Table 5-22.)

Table 5-22 Hanging paragraph macro

Type	Form	Explanation
Macro	.XP	Paragraph with the first line exdented by the value of register P I (5 ens). The paragraph is offset vertically by value of register P D (1v).

Creating a quote paragraph

In ms, a quote paragraph is indented 5 ens from both the left and right margins. Subsequent text is centered and generated as an offset block. (See Table 5-23.)

Table 5-23 Quote paragraph macro

Type	Form	Explanation
Macro	.QP	Quote paragraph. The paragraph is centered and indented left and right by the value of Register Q I (5 ens) and offset vertically by the value of register P D (1v).

Changing the spacing between paragraphs

The default distance between paragraphs is one vertical space. To change this value in ms, reset register PD. (See Table 5-24.)

Table 5-24 Paragraph spacing register

Type	Form	Explanation
Register	.PD	Paragraph distance Initial value: 1v in nroff, 0.3v in troff

Creating headings

Two types of section headings are available with these macro packages: unnumbered and numbered. In both cases, the heading is on the left margin and is preceded by one blank line, and the text of the section is immediately following the heading (without a blank line). In `troff` the heading is printed in boldface; in `nroff` it is underlined. A paragraph macro must follow the heading macro if a vertical space or indentation is desired.

Creating numbered headings

In `ms`, the .NH macro produces automatically numbered section headings. (See Table 5-25.) An optional level number indicates a subsection from 1 to 5. For example,

```
.NH 1
First-level heading
.LP
```
text
```
.NH 2
Second-level heading
.LP
```
text

produces

1. First-level heading
text
1.1 Second-level heading
text

Table 5-25 Numbered headings macro

Type	Form	Explanation
Macro	`.NH [x]`	Begin automatically numbered heading, where x is the heading level number (1 through 5). If $x = 0$, numbering is reset to 0.

Working with unnumbered headings

The `ms` macro `.SH` produces section headings that are not numbered. (See Table 5-26.)

Table 5-26 Unnumbered headings macro

Type	Form	Explanation
Macro	`.SH`	Begin left-adjusted section heading, separated from the preceding text by one vertical space.

Creating page headers and footers

Text printed at the top of each page is called a page header. Text printed at the bottom of each page is called a page footer. You can specify three separate headers and footers (left, right, and center) using either string registers or macros.

In `ms`, six string registers set up the standard layout of headers and footers. Those registers that do not have predetermined default values are set with the following command:

`.ds` *register-name text*

For example, to print the word "DRAFT" in the lower right corner of every page of a document, define register `RF` (right footer) as follows:

```
.ds RF DRAFT
```

To clear the register, use this command:

```
.rm RF
```

You can use these macros to create custom headers and footers that appear on even or odd pages. Arguments to these macros must be enclosed within a set of four apostrophes indicating placement on the line within three fields (left, right, and center), for example,

`.OH` *'left'center'right'*

The `ms` macro package has many other ways of dealing with headers and footers. The next sections explain these macros.

Using standard headers

Use the following string registers to store text put in the left, center, and right headers. Only the center header (register CH) contains a default string. In both nroff and troff, unless specified otherwise, register CH contains the current page number surrounded by hyphens. If you don't want a centered page number, you can easily remove it or move it to another position. The remaining fields must be set manually. (See Table 5-27.)

Table 5-27 Standard header macros

Type	Form	Explanation
Register	.LH	Left header
Register	.CH	Center header
		Initial value: current page number surrounded by hyphens
Register	.RH	Right header

Using standard footers

Use the following string registers to store text put in the left, center, and right footers. In nroff, the center footer (CF) contains the current date as the default string. In troff this field is empty. (See Table 5-28.)

Table 5-28 Standard footer macros

Type	Form	Explanation
Register	.LF	Left footer
Register	.CF	Center footer
		Initial value: current date (nroff only)
Register	.RF	Right footer

Customizing headers and footers

You can specify headers and footers on even- and odd-numbered pages by defining macros .EH, .OH, .EF, and .OF. (See Table 5-29.)

For example, if you want the title of your document to be in the left footer on even-numbered pages and in the right footer on odd-numbered pages, use the following commands:

.EF *' titl' ' '*

.OF *' ' ' title'*

Table 5-29 Customized header and footer macros

Type	Form	Explanation
Macro	.EF[*' l' c' r*]	Print page footer only on even pages. The three strings specified between the four apostrophes indicate left, center, and right. When used without arguments, cancel previously specified even footer.
Macro	.EH[*' l' c' r'*]	Print page header only on even pages. The three strings specified between the four apostrophes indicate left, center, and right. When used without arguments, cancel previously specified even header.
Macro	.OF[*' l' c' r*]	Print page footer only on odd pages. The three strings specified between the four apostrophes indicate left, center, and right. When used without arguments, cancel previously specified odd footer.
Macro	.OH[*' l' c' r*]	Print page header only on odd pages. The three strings specified between the four apostrophes indicate left, center, and right. When used without arguments, cancel previously specified odd header.

Printing a header and/or footer on the first page

By default, the printing of headers and footers begins on page two of your document. To print a header, a footer, or both on page one of your document, use the `ms` macro `.P1`; this will print whatever is defined as your header or footer in the registers or in the macros. It must be used before the beginning of the text. (See Table 5-30.)

Table 5-30 Printing header/footer on first page macro

Type	Form	Explanation
Macro	`.P1`	Print header and/or footer on the first page of the document. Must be placed at the beginning of the text

Creating multiline headers and footers

The `.PT` (page title) and `.BT` (bottom title) commands are used to define macros for multiline page headers and footers. Define this macro at the beginning of your file, for example,

```
.de PT (or .BT)
.tl 'left' center' right'
.tl 'left' center' right'
..
```

If you need a three-line header or footer, add the formatting instruction

```
'sp-1
```

before the first `.tl` instruction so the header lines will begin one line higher on the page. Make sure you use an apostrophe (`'`) and not a period (`.`) with the `'sp-1` instruction. (See Chapter 3, "`nroff/troff` Formatters," for a full explanation of the difference between the apostrophe and the period in `troff` requests.)

After you have defined these macros, the system automatically uses the new definition when a page is begun.

Setting title length

Register LT determines horizontal distance available for headers and footers. By default, it is equal to the line length (LL). (See Table 5-31.)

Table 5-31 Setting title length register

Type	Form	Explanation
Register	.LT	Total length of headers and footers Initial value: 6 inches (or the same as register LL)

Keeping text together on a page

The ms macro package provides commands to keep a block of text together on one page. There are two ways to do this: the standard (or static) keep and the floating keep.

Forcing a page with static keeps

In ms, the **static keep** begins with .KS and ends with .KE. If the number of lines within these two macros exceeds the remaining lines on the page, a page break is forced, and the material in the block is printed on the next page. (See Table 5-32.)

Table 5-32 Static-keeps macros

Type	Form	Explanation
Macro	.KS	Begin static keep.
Macro	.KE	End static or floating keep.

Using floating keeps

In ms, the **floating keep** begins with .KF and ends with .KE. If the number of lines in a block of text exceeds the remaining lines on the page and it is necessary to force a page break, the regular text material continues to print until it reaches the end of the page, and the block of text is printed. It differs from a static keep in that it waits for a natural page break rather than forcing one. (See Table 5-33.)

Table 5-33 Floating-keeps macros

Type	Form	Explanation
Macro	.KF	Begin floating keep.
Macro	.KE	End static or floating keep.

Indenting blocks of text

ms has facilities for indenting blocks of text to the right. The .RS macro shifts the text to the right. The default value for the shift is 5 ens, but this can be changed by resetting number register PI.

You can use more than one .RS to increase the amount of indentation. The only limit is the right margin. For each .RS entered, you must enter a .RE to cancel it. For example, if you enter five .RS macros, you must enter five .RE macros. (See Table 5-34.)

Table 5-34 Right-shift macros

Type	Form	Explanation
Macro	.RS	Right shift. Move indentation to the right by the value of register PI (5 ens). .RS can be nested.
Macro	.RE	End right shift. Move indentation to the left by the amount that the corresponding .RS is moved to the right.

Creating displays

Displays format text without filling or adjusting. Several types of displays are available with ms, both those with keep and those without. ms displays keep text on a single page. If you don't want the text to be kept on a single page, use the ms displays without keep (.ID, .LD, .CD, .BD).

Using ms displays

If you want text to be kept on a single page, use the standard ms display (.DS [x]), where x can designate left, right, centered, or block.

Standard display format

A standard display is automatically put into a keep (see "Keeping Text Together on a Page" earlier in this chapter). A standard display can be indented, left-adjusted, centered, or in block format, depending on the argument you use. (See Table 5-35.)

Table 5-35 Standard display macros

Type	Form	Explanation
Macro	.DS [xy]	Start displayed text. The text is formatted without filling or adjusting. x determines the format of the display:
		$x=$ I Indented
		$x=$ L Left-adjusted
		$x=$ C Each line centered
		$x=$ B Lines centered as a group
		y sets the amount (in ens) of indentation (the default is the value of register P I, or 5 ens).
Macro	.DE	End display.

Indented display

The indented display is the same as .DS I except that it does not invoke a keep.
Displayed material is formatted 5 ens to the right of the left margin (or the value of
register P I). (See Table 5-36.)

Table 5-36 Indented display macro

Type	Form	Explanation
Macro	.ID	Indented display (no keep) Initial value: amount of register P I (5 ens)

Left-adjusted display

The left-adjusted display is the same as .DS L except that it does not invoke a keep.
Displayed text is formatted in a block at the left margin. (See Table 5-37.)

Table 5-37 Left-adjusted display macro

Type	Form	Explanation
Macro	.LD	Left-adjusted display (no keep)

Centered display

The centered display is the same as .DS C except that it does not invoke a keep. Each
line of text is centered individually. (See Table 5-38.)

Table 5-38 Centered display macro

Type	Form	Explanation
Macro	.CD	Centered display (no keep). Lines are centered individually.

Block display

The block display is the same as `.DS B` except that it does not invoke a keep. Displayed text is centered and left adjusted as a group. (See Table 5-39.)

Table 5-39 Block display macro

Type	Form	Explanation
Macro	`.BD`	Left-adjusted then centered display (no keep)

Display distance

Display distance is the amount of vertical space surrounding a display. The default settings are one vertical space (`nroff`) or one-half vertical space (`troff`). It is set with register `DD`. (See Table 5-40.)

Table 5-40 Display distance macro

Type	Form	Explanation
Register	.DD	Vertical distance surrounding displays Initial value: one vertical space in `nroff`; one-half vertical space in `troff`

Producing tables and equations

The following `ms` macros are used with the system preprocessors `tbl` and `eqn` to produce tables and equations. For complete instructions on using these programs, refer to Chapter 7, "`tbl` Tables," and Chapter 8, "`eqn` Equations."

Creating tables

Text placed between the delimiters .TS and .TE (table start and table end) are processed by the table formatting program tbl. If you are producing a multipage table and you want a standard heading to be printed on each page of the table, you should use the .TS H form of the command. Type .TH at the end of the heading material. For example,

```
.TS H
center tab(:) ;
c c .
.TH
```
heading: data
table: data
table: data
```
.TE
```
repeats the heading

heading data

on every page. (This is a feature not of tbl but of ms.) (See Table 5-41.)

Table 5-41 Table macros

Type	Form	Explanation
Macro	.TS [H]	Table start. Supplies one-half vertical space between preceding text and table. Argument H indicates that the material that follows (until a .TH) is heading text to be repeated for multipage tables.
Macro	.TE	Table end.
Macro	.TH	End table heading. Used only with the .TS H macro.

Creating equations

Text placed between the delimiters `.EQ` and `.EN` (equation start and equation end) are processed by the equation formatting program `eqn`. (See Table 5-42.)

You must use displays with keep (`.DS` / `.DE`) around displayed equation specifications. (See Chapter 8, "eqn Equations," for a discussion of the difference between displayed and inline equations.)

By default, displayed equations are centered. You can specify the placement (centered, left adjusted, or indented) by supplying the appropriate argument to the `.EQ` command. Following this argument, you can also specify an equation number to label the equation. The label is generated at the right margin. For example,

```
.DS
.EQ (1)
sum from i=0 to {i = inf} x sup i 4 over pi
.EN
.DE
```

produces

$$\sum_{i=0}^{i=\infty} x^i \frac{4}{\pi}$$

Table 5-42 Equations macros

Type	Form	Explanation
Macro	`.EQ` [xy]	Begin equation. Output preceded by one vertical space and automatically centered. x controls the placement of the equation: $x =$ `L` Left-adjusted $x =$ `I` Indented $x =$ `C` Centered Argument y supplies an equation number and prints it at the right margin.
Macro	`.EN`	End equation.

Creating footnotes

You can produce footnotes with ms by placing text between footnote start and end macros, .FS and .FE. The material is collected, saved, and printed at the bottom of the current page. The footnote is printed two points smaller than the text and is separated from the main body of text by a horizontal line. (See Table 5-43.)

You can produce footnotes that are numbered automatically by placing the string ** immediately following the text to be footnoted. ms also permits you to define your own footnote label. For example, if you want your footnotes to be labeled alphabetically, you can enter the following text:

```
.LP
This is the sentence I am referencing [A].
.FS
[A] This is the text of the footnote.
.FE
```

Table 5-43 Begin and end footnote macros

Type	Form	Explanation
Macro	.FS	Begin footnote.
Macro	.FE	End footnote.

Changing footnote style

In a standard footnote, a label is printed as a superscript, and the first line is indented. You can suppress both these features with ms, as well as produce the footnote text as an indented paragraph. Use the number register FF to modify the default format of a footnote. (See Table 5-44.)

Table 5-44 Footnote format register

Type	Form	Explanation
Register	FF x	Footnote format
		$x = 1$ Suppress superscripting of footnote label.
		$x = 2$ Suppress indentation of first line of footnote text.
		$x = 3$ Footnote as indented paragraph.

Changing footnote indent

In ms, the footnote indent register is used to change a footnote's distance from the left margin. The default is two ens. (See Table 5-45.)

Table 5-45 Footnote indent register

Type	Form	Explanation
Register	.FI	Footnote indent. Controls the amount of indentation from the left margin.
		Initial value: 2 ens

Changing footnote length

By default, the length of a footnote is 5.5 inches, just slightly shorter than the default line length. You can change this value with ms by resetting register FL. (See Table 5-46.)

Table 5-46 Footnote length register

Type	Form	Explanation
Register	.FL	Footnote line length
		Initial value: 5.5 inches

Using references

You can classify books, journal articles, book chapters, and reports with the ms macros `.]-`, `.[0`, and `.[n`. (See Table 5-47.) They must be used in conjunction with the troff preprocessor refer. See refer(1) in *A/UX Command Reference*.

Table 5-47 Reference macros

Type	Form	Explanation
Macro	`.]-`	Begin refer reference.
Macro	`.[0`	End of unclassifiable reference.
Macro	`.[n`	Classifiable reference:

$n = 1$ journal article
$n = 2$ book
$n = 3$ book article
$n = 4$ report

Creating an index or a table of contents

You should enclose all entries you want placed in an index in the ms begin and end delimiters `.XA` and `.XE`. Additional entries are preceded by the macro `.XA`.

These macros can be used throughout the body of text in combination with section heading macros to automatically generate a table of contents with page numbers, for example,

`.SH`
heading-text
`.XS 1`
heading-text
`.XE`

If you are using numbered headings and you want these numbers included in the table of contents, use this format:

```
.NH
```
heading-text
```
.XS 1
```
*(SN *heading-text*
```
.XE
```

The *(SN string will be replaced with the number of the heading when the table of contents or index is printed.

The final output of the index or table of contents is produced with either the .PX or the .TC macro. The difference between these macros is that .TC prints a centered "CONTENTS" heading at the top of the page and page numbering is reset to Roman numerals (as in this document).

Understanding index format

Material to be printed in an index or table of contents should be placed between the .XS and .XE macros. Use the .XA macro for additional ms entries:

```
.XS
```
index-entry
```
.XA
```
additional-index-entry
```
.XA
```
additional-index-entry
```
.XE
```

You can designate the page number of the indexed material as an argument to .XS or .(x:

```
.XS
```
 page-number

```
.(x
```
 page-number

You can change the indentation level by assigning a value to the argument following the page number:

```
.XS 1 5
```

(See Table 5-48.)

Table 5-48 Index format macros

Type	Form	Explanation
Macro	.XS [x y]	Begin index entry, where x is the page number of the entry and y is the amount of indentation (in ens).
Macro	.XA [x y]	Additional index entry, where x is the page number of the entry and y is the amount of indentation (in ens).
Macro	.XE	End index entry.

Printing the index

The .PX macro is used to print a formatted list of the text items designated by the index macros. (See Table 5-49.)

Table 5-49 Index print macro

Type	Form	Explanation
Macro	.PX	Print index.

Printing the table of contents

The .TC macro prints a list of the text items designated by the index macros. (See Table 5-50.) It differs from the .PX macro described above in two ways:

- It provides a centered heading ("CONTENTS") at the top of the page.
- It resets page numbering to lowercase Roman numerals.

Table 5-50 Table of contents print macro

Type	Form	Explanation
Macro	.TC	Print table of contents. Preceded by page break, and the page numbering is reset to lowercase Roman numerals.

Drawing boxes

You can draw a box around a single word or a group of words with the box macros.

Boxing a word

Use the `.BX` command to draw a box around a single word. (See Table 5-51.) The word to be boxed is entered as an argument to the macro, for example,

`.BX` *word*

Table 5-51 Boxed word macro

Type	Form	Explanation
Macro	`.BX` *x*	Draw a rectangular box around a word, where *x* is any word.

Boxing a block of text

You can draw a box around a group of words with the `.B1` and `.B2` macros. Text to be boxed is entered on the line following the `.B1`. (See Table 5-52.)

Table 5-52 Boxed block of text macros

Type	Form	Explanation
Macro	`.B1`	Begin boxed text.
Macro	`.B2`	End boxed text.

Checking your work

You can check your file for formatting errors with the `checknr` program. `checknr` examines your file and reports any unrecognized macros or unbalanced macro constructions. For example, it will find any `.DS` commands that are not terminated with `.DE`, and it will verify that each `.RS` command has a corresponding `.RE` command.

To run `checknr`, enter the command

`checknr` *file*

Any discrepancies are written to the standard output. Or, if you prefer, you can direct the output from `checknr` to a file so you can examine it later:

`checknr` *file* `>` *output-file*

For more detailed instructions on using this program, refer to `checknr`(1) in *A/UX Command Reference.*

Using `nroff/troff` commands in `ms`

The `ms` macro package was designed to meet most text-processing needs, making it unnecessary for users to learn the details of the complicated `nroff/troff` formatting language. However, you can use `nroff/troff` commands in conjunction with the `ms` macros without losing the benefits and simplicity of using a macro package.

In addition to the `.nr` and `.ds` commands used to define number and string registers, you can use the following `nroff/troff` commands in a file processed with the `ms` macros:

`.ce` *n*	Center *n* lines of text. If *n* is omitted, only the line following is centered.
`.sp` *n*	Print *n* blank lines. If *n* is omitted, one blank line is printed.
`.br`	Start a new output line.
`.bp`	Begin a new page.
`.pl` *n*	Set the page length to *n*.
`.ls` *n*	Set the line spacing to *n*.
`.na`	No adjust. Turns off right-margin justification.
`.ad`	Adjust both margins.

Note that you can use `.ls`, `.na`, and `.ad` anywhere in your document; the remaining requests, however, must not appear until after the initializing macro (see "Sequence of Beginning Macros" earlier in this chapter).

Creating your own macros

You can create your own macro out of a sequence of `nroff/troff` commands and other defined macros. The basic procedure is to set up a definition string and then name the macro. Because `ms` uses uppercase macro names, it is probably a good idea to use `ms` custom macro names that are either a single lowercase letter or an uppercase letter followed by a lowercase letter.

Conventions used in this reference

The following conventions are used to describe macro names:

n	Digit
x	Alphanumeric character

All other characters are literals (characters that stand for themselves).

Macro and string names are kept in a single internal table. Therefore, there must be no duplication among such names. Number register names are kept in a separate table.

Format of names used by ms

Macros are in the form *x*, *xx*, or *xn* (for example, macros .I, .PP, and .P1), and registers are in the form *xx* (for example, PO).

Names used by eqn/neqn and tbl

The mathematical equation preprocessors, eqn and neqn, use registers and string names of the form *nn*. The table preprocessor, tbl(1), uses T&, T#, and TW, and names of the form

$x- \quad x+ \quad x| \quad {}^{\wedge}x \quad {}^{\wedge}x \quad \#x$

Reference tables

Tables 5-53 through 5-55 provide summaries of the macros, number registers, and strings used in ms.

Table 5-53 ms macro summary

Name	Description
.AB	Begin abstract.
.AE	End abstract.
.AI	Author's institution.
.AU	Author's name.
.B [*x*]	Print *x* in boldface. If *x* is not present, print all subsequent text in boldface.
.B1	Begin boxed text.
.B2	End boxed text.
.BD	Block display; center entire block.
.BT	Bottom title, printed at foot of page.
.BX [*x*]	Print *x* in a box.
.CD	Centered display (no keep).

Table 5-53 ms macro summary (*continued*)

Name	Description
.CT	Chapter title. Page number is moved to register CF (thesis mode only).
.DA [x]	Print current date at the bottom of each page. With a date as an argument, uses that date in place of the current date.
.DE	End display.
.DS [x y]	Begin display with keep:

    $x =$ I   Indented

    $x =$ L   Left-adjusted

    $x =$ C   Centered

    $x =$ B   Block

    $y =$ amount of indentation

Name	Description
.EF [' l' c' r']	Even footer.
.EH [' l' c' r']	Even header.
.EN	End equation.
.EQ [x y]	Begin equation:

    $x =$ I   Indented

    $x =$ L   Left-adjusted

    $x =$ C   Centered

    $y =$ E   Equation label

Name	Description
.FE	End footnote.
.FS [x]	Start footnote, where x is a user-defined label.
.I [x]	Print x in italic. If x is not present, print all subsequent text in italic.
.ID	Indented display (no keep).
.IP [x y]	Indented paragraph, where x is a label and y is the indentation.
.KE	End keep (static or floating).
.KF	Begin floating keep. Name description
.KS	Begin static keep.
.LD	Left-adjusted display (no keep).
.LG	Increase point size by 2.
.LP	Left-block paragraph.

(continued)➡

Table 5-53 ms macro summary (*continued*)

Name	Description
.MC [*x y*]	Print text in multiple columns, where *x* is the column width and *y* is the gutter width.
.ND [*x*]	Suppress printing of date in page footer, where *x* is the date on the cover sheet (released paper format).
.NH [*x*]	Begin numbered heading, where *x* is the heading level. If *x* = 0, level is reset to 0.
.NL	Return point size to normal. Initial value: 10
.OF [' *l'* *c'* *r'*]	Odd footer.
.OH [' *l'* *c'* *r'*]	Odd header.
.P1	Print header (including page number) on the first page.
.PP	Standard paragraph with the first line indented.
.PT	Page title, printed at the head of each page.
.PX [no]	Print index. no suppresses the title.
.QP	Quotation mark paragraph, centered and indented 5 ens from both margins.
.R	Return to roman font.
.RE	End right shift.
.RP [no]	Begin released-paper format. no suppresses the title on the first page.
.RS	Begin right shift; start relative indentation.
.SH	Begin unnumbered section heading, left-adjusted and boldfaced.
.SM	Decrease point size by 2.
.TA *x*	Set tabs to *x*, where *x* is the number of ens. Initial value: increments of 5 ens.
.TC [no]	Print table of contents. no suppresses the title.
.TE	End table.
.TH	End running table heading.
.TL	Print centered title in boldface 2 points larger.
.TM	Thesis mode.
.TS [H]	Begin table. H indicates a multipage header.
.UL *x*	Underline *x*.
.UX	UNIX trademark message.
.XA [*x y*]	Additional index entry. *x* is the page number of the entry and *y* is the amount of indentation (in ens).

Table 5-53 ms macro summary (*continued*)

Name	Description
.XE	End index entry.
.XP	Exdented paragraph.
.XS [*x y*]	Begin index entry. *x* is the page number of the entry and *y* is the amount of indentation (in ens).
.1C	Resume one-column printing.
.2C	Begin two-column printing.
.]-	Begin reference.
.[0	End unclassifiable reference.
.[*n*	Classifiable reference.
	n = 1 Journal article
	n = 2 Book
	n = 3 Book article
	n = 4 Report

Table 5-54 Number register summary

Name	Description
CF	Center footer.
	Initial value: current date (nroff only)
CH	Center header.
	Initial value: current page number surrounded by hyphens
DD	Display distance.
	Initial value: 1v in nroff, .5v in troff
FF [*x*]	Footnote format.
	x = 1 Suppress superscripting of footnote label.
	x = 2 Suppress indentation of first line of footnote text.
	x = 3 Footnote as indented paragraph.
	Initial value: 0

Table 5-54 Number register summary (*continued*)

Name	Description
FI	Footnote indent.
	Initial value: 2 ens
FL	Footnote length.
	Initial value: 55i
FM	Footer margin.
	Initial value: 1i
HM	Header margin.
	Initial value: 1i
LF	Left footer.
LH	Left header.
LL	Line length.
	Initial value: 6i
LT	Title length.
	Initial value: same as `LL` (6i)
PD	Paragraph distance.
	Initial value: 1v in `nroff`, 03v in `troff`
PI	Paragraph indent.
	Initial value: 5 ens
PO	Page offset.
	Initial value: 0 in `nroff`), ~1i in `troff`
PS	Point size.
	Initial value: 10
RH	Right header.
QI	Quotation mark paragraph indent.
	Initial value: 5 ens
VS	Vertical spacing.
	Initial value: 12v 21

Table 5-55 ms string summary

Name	Description
\ * '	Acute accent (before letter)
\ * *	Automatically numbered footnote
\ * ,	Cedilla (before letter)
\ * ^	Circumflex (before letter)
\ *_	Dash (-- in nroff, – in troff)
\ * (DY	Day (current date)
\ * `	Grave accent (before letter)
\ * (MO	Month
\ *Q	Quotation mark (" in nroff, " in troff)
\ * ~	Tilde (before letter)
\ * :	Umlaut (before letter)
\ *U	Unquotation mark (" in nroff, " in troff)

6 me Macros

This chapter is a reference for the me macro package. It's a good idea to skim this chapter for a general understanding of the me macro package and then read specific sections in detail as needed.

What are me macros?

me is a collection of text-formatting macros for the A/UX text formatters nroff and troff. It was designed for writing thesis papers at the University of California at Berkeley. Some features of me are not available in ms, so A/UX Release 3.0 supports both packages. You can use only one of these packages at a time, however, so you may wish to read this chapter and make a decision about which package to use before you actually begin formatting a document.

For a complete discussion of text formatting concepts and principles, refer to Chapter 1, "Introduction to A/UX Text Processing."

How input is read

Formatters fill output lines from one or more input lines. You can justify output lines so that both the left and right margins are aligned. As lines are being filled, words may also be hyphenated as necessary. You can turn any of these modes on and off (with the .na, .ad, .hy, .nf, and .fi formatter requests; turning off fill mode also turns off justification and hyphenation). Certain formatting commands (requests and macros) stop filling the current output line, print the line (of whatever length), and begin subsequent text on a new output line. This printing of a partially filled output line is called a break. A few formatter requests cause a break.

Understanding arguments and double quotation marks

In me, you can use an argument to modify a macro. For example, the me macro .pp begins a standard paragraph.

Any macro argument containing ordinary (paddable) spaces must be enclosed in double quotes. A double quotation mark is a single character that must not be confused with two apostrophes, acute accents, or grave accents. If an argument containing such spaces is not enclosed in double quotation marks, it will be treated as several separate arguments.

Sequence of beginning macros

Text files processed by the me macros must begin with one of the following macros:
`.pp`, `.lp`, `.ip`, `.np`, `.sh`, and `.uh`.

These macros initialize the file and must precede a break caused by blank lines, leading spaces, or `.sp`, `.br`, and `.ce` `troff` requests.

Using basic document formats

The me macro package has facilities for formatting the basic elements of a document, such as the cover page, margins, and spacing.

Title pages

There are no headers or footers on a title page, and unlike other pages, you are allowed to leave blank space by spacing down from the top. The `.tp` macro produces a title page. (See Table 6-1.)

Table 6-1 Title pages macro

Type	Form	Explanation
Macro	`.tp`	Print title page.

Chapter titles

The `.+c` *T* macro can be used to start chapters in me. Each chapter is automatically numbered from one, and a heading is printed at the top of each chapter with the chapter number and the name *T*. This information is moved to the footer of the first page of the chapter. If the name *T* is not specified, the output is a chapter with no heading. (See Table 6-2.)

Although a document's preliminary sections—the abstract, table of contents, and so on—are normally placed at the beginning, you should format and print them last when using me. This is so that index entries can be collected and then printed for the table of contents. At the end of the document's main text, you can use the .++P macro, which begins the preliminary sections. After issuing this request, you can use the .+c macro to begin one preliminary section of the document and print the page numbers in lowercase roman numerals.

You can use the .+c macro repeatedly to begin different preliminary sections, such as the abstract, table of contents, and acknowledgments. Then you can use the .++B macro to begin the bibliography at the end of the document. You will have to rearrange the document physically after printing to place the preliminary sections at the beginning.

Table 6-2 me chapter titles macros

Type	Form	Explanation
Macro	.+c [T]	Print chapter heading, where T is the name of the chapter. If T is omitted, the chapter page is printed with no heading.
Macro	.++P	Print preliminary section of paper with lowercase roman numeral page numbers.
Macro	.++B	Print the bibliographic section of the paper.

Thesis format

The .th macro sets up the headers, footers, margins, and spacing of the formatter to format a thesis according to the rules established at the University of California at Berkeley. The correct headers, footers, margins and spacing are set up. (See Table 6-3.)

Table 6-3 Thesis format macro

Type	Form	Explanation
Macro	.th	Set up Berkeley thesis format.

Changing the look of the document

A document formatted with the me macros is produced in a standard page layout. By default, text is generated in a single column, and a line of text is 6 inches from margin to margin. The left margin is 1 inch (in troff) from the edge of the paper, point size is set to 10 points, vertical space is set to 12 points, and tab stops are set every 5 spaces. The following macros and number registers permit you to change these default features and customize your page layout. You can also change fonts and remove the date.

Creating multicolumn output

Output from troff is normally a single column of text. Placing the me command .2c in your file causes the output to be printed in two-column format. Each column is printed with a width of 7/15 of the current line length and the gap between the two columns is 1/15 of the full line length.

To print text in more than two columns,you can request a new column with .bc. To revert back to single column output, use .1c.

The number of columns is computed automatically, based on the maximum number of columns of the specified width that can fit within the current line length. The column width argument must be numeric, and unless indicated otherwise, the unit of measurement is assumed to be in ens.

The *gutter-width* argument permits you to control the distance between columns.

Any change in the number of columns specified (except from one to two or greater) causes a page break. (See Table 6-4.)

Table 6-4 Multiple column macros

Type	Form	Explanation
Macro	.2c	Print text in two equal columns.
Macro	.bc	Begin new column.
Macro	.1c	Restore single-column output.

Setting point size and vertical spacing

Number registers are used to set default point size and vertical spacing. In me there is a register for paragraph point size, .nr pp, a register for section header point size, .nr sp, and a register for title point size called .nr tp. (To change relative point size using macros, see "Changing the String Point Size" later in this chapter). The default point size for regular text is 10 points, and the default point size for footnotes is 8 points. The two-point difference allows for adequate spacing between lines.

The vertical spacing is set to be proportional to the type size. (See Table 6-5.)

Table 6-5 Point size and vertical spacing registers

Type	Form	Explanation
Register	.nr pp	Paragraph point size
		Initial value: 10
Register	.nr sp	Section heading point size
		Initial value: 10
Register	.nr tp	Title point size
		Initial value: 10

Changing top and bottom margins

By default, the distance between the header and footer text and the top and bottom edges of the paper is one inch.

Changing line length

The default length of a line of text is six inches from left to right margin.

Changing page offset

The position of the left margin is determined by two dimensions: page offset and indentation. Indentation controls the current left margin, whereas page offset controls the absolute left margin.

Page offset is the distance betweeen the left margin and the left edge of the paper. Indentation is expressed as a distance to the right of page offset. You can change indentation within your document (see "Indenting Paragraphs" later in this chapter), but page offset is defined at the beginning of your document and usually remains constant throughout.

The default page offset is 1 in `troff` and 0 in `nroff`.

Changing fonts

You can use the following macros to emphasize words or groups of words. (See Table 6-6.) Typewritten or line-printed material is usually emphasized with underlining. Typeset and typeset-quality material is emphasized with **boldface** or *italics.*

In me, use `.b` for bold and `.i` for italics. There are several ways of using these macros in your text.

In me, `.b` or `.i` can be followed by a single word. In that case, only that word is emphasized. The macros for boldfacing and italicizing can be followed by a group of words on the same line. These must be enclosed in double quotation marks.

The underline macro applies only to text processed with `troff`. It underlines one word at a time. If multiple word underlining is desired, you must enter individual underlining commands for each word. Enclosing multiple words in quotes does not work. For example, in me you could use

text

`.u` *word1*
`.u` *word2*
`.u` *word3*

Table 6-6 Font changing macros

Type	Form	Explanation
Macro	.b [x]	Print x in boldface (troff only).
Macro	.i [x]	Print x in italics.
Macro	.u x	Underline x (troff only).

Changing the string point size

In me there is only one .sm macro to set a word or phrase in a smaller point size (see Table 6-7). This macro is used for temporary size changes for a single word or a small group of words. (See "Setting Point Size and Vertical Spacing" earlier in this chapter to change absolute point size.)

Table 6-7 String point size changing macro

Type	Form	Explanation
Macro	.sm	Decrease point size by 2.

Structuring the page

Using me macros, you can create indented and labeled paragraphs, establish headings and change their appearance, create customized headers and footers, and control page breaks to create the layout that best suits your purposes.

Creating paragraphs

The me macro package provides several commands that determine the style of your paragraph. In all cases, the formatter skips one vertical space before generating the text of the paragraph.

Creating the standard paragraph

The first line of a standard paragraph is indented. All other lines are generated at the left margin. The default indentation is 5 ens, but can be changed by setting the number register P I (see "Indenting Paragraphs" later in this chapter.) (See Table 6-8.)

Table 6-8 Standard paragraph macro

Type	Form	Explanation
Macro	.pp	Standard paragraph

Creating a left-block paragraph

The text of a left-block paragraph is generated as a left-adjusted block. (See Table 6-9.)

Table 6-9 Left-block paragraph macro

Type	Form	Explanation
Macro	.lp	Left-block paragraph

Indenting paragraphs

All lines of an indented paragraph are indented a certain value. (See Table 6-10.)
In me, the command .ip can be used in three ways:

```
.ip
.ip label .
.ip label value
```

The first example produces a basic indented paragraph. Text is generated as a block five spaces from the left margin.

The other two forms of the indented paragraph command permit you to label your paragraph with some alphanumeric character. These can be used to produce numbered or bulleted lists. For example,

```
.ip (1)
This is a labeled indented paragraph.
```

produces

(1) This is a labeled indented paragraph.

You can substitute any character for the number. For example,

```
.ip *
This is a labeled indented paragraph.
```

produces

* This is a labeled indented paragraph.

You can also assign a value for the indentation level:

```
.ip (1) 10
```

Instead of the default indentation (5 ens), the formatter now indents the text 10 ens.

Table 6-10 Indented paragraph macros

Type	Form	Explanation
Macro	.ip [xy]	Indented paragraph, where x is the label, and y is the indentation. Default indentation is 5 ens. The number register for setting paragraphg indentation is ii.
Macro	.np	Numbered indented paragraph. The numbering is reset at the next .pp, .lp, or .sh.

Table 6-11 Indented paragraph register

Type	Form	Explanation
Register	.i	Paragraph indentation. The values are unscaled and are read as ens.

Creating headings

Two types of section headings are available with the me macro package: unnumbered and numbered. In both cases, the heading is on the left margin and is preceded by one blank line, and the text of the section is immediately following the heading (without a blank line). In troff the heading is printed in boldface; in nroff it is underlined. A paragraph macro must follow the heading macro if a vertical space or indentation is desired.

Creating numbered headings

In me, the .sh macro produces automatically numbered section headings. (See Table 6-12.) An optional level number indicates a subsection from 1 to 5. For example,

```
sh 1 First-level heading
.lp
text
.sh 2
Second-level heading
.lp
text
```

produces the output

1. First-level heading
text
1.1 Second-level heading
text

    me also has the ability to indent sections proportionally to the depth of the section.

The command

```
.nr si Nx
```

will cause each section to be indented by N. N must have a scaling factor of the form x, where x is the unit N is measured in. The most common units are i for inches, c for centimeters, and n for ens (the width of a single character).

Table 6-12 Numbered headings macros

Type	Form	Explanation
Macro	`.sh` [x]	Begin automatically numbered heading, where *x* is the heading level. Numbering is reset at new paragraph request. (The argument number can also be placed after the title of the section, as in `.sh "title" 1.`)
Macro	`.si` [Nx]	Indented section heading. Indents each section by *N* in *x* units in the section number.

Working with unnumbered headings

The me macro `.uh` produces section headings that are not numbered. (See Table 6-13.)

Table 6-13 Unnumbered headings macro

Type	Form	Explanation
Macro	`.uh`	Begin left-adjusted section heading, separated from the preceding text by one vertical space.

Creating page headers and footers

Text printed at the top of each page is called a page header. Text printed at the bottom of each page is called a page footer. You can specify three separate headers and footers (left, right, and center) using either string registers or macros.

In me, simple requests handle headers and footers. They are three-part titles, as in ms, and a percent sign is used for the current page number:

```
.he "%"
.fo "Successful Author""My Story"
```

This will produce output with the current page number centered at the top of each page, "Successful Author" in the lower left corner, and "My Story" right justified in the lower right corner.

Keeping text together on a page

The me macro package provides commands to keep a block of text together on one page. There are two ways to do this: the standard (or static) keep and the floating keep.

Forcing a page with static keeps

In me, the static keep is accomplished with the block macros `. (b` and `.) b`. If the number of lines within these two macros exceeds the remaining lines on the page, a page break is forced, and the material in the block is printed on the next page. (See Table 6-14.)

Table 6-14 Static keeps macros

Type	Form	Explanation
Macro	. (b	Begin static keep.
Macro	.) b	End static keep.

Using floating keeps

In me, the **floating keep** is accomplished with the `. (z` and `.) z` macros. If the number of lines in a block of text exceeds the remaining lines on the page and it is necessary to force a page break, the regular text material continues to print until it reaches the end of the page, and the block of text is printed. It differs from a static keep in that it waits for a natural page break rather than forcing one. (See Table 6-15.)

Table 6-15 Floating keeps macros

Type	Form	Explanation
Macro	. (z	Begin floating keep.
Macro	.) z	End floating keep.

Indenting blocks of text

me has facilities for indenting and centering blocks of text.

Centering blocks of text

Sometimes you may want to center several lines of text as a group, rather than centering each line separately. This can be accomplished with the `.(c` and `.)c` macros. All the lines between these macros will be centered as a unit, with the longest line centered on the page and the rest of the lines centered around it. Centered blocks are *not* keeps, but you may use them inside keeps. (See Table 6-16.)

Table 6-16 Centering macros

Type	Form	Explanation
Macro	`.(c`	Begin centered block.
Macro	`.)c`	End centered block.

Creating displays

Displays format text without filling or adjusting. Several types of displays are available with me, both those with keep and those without. me displays allow text to cross page boundaries.

Using me displays

If you don't want the text to be kept on a single page, use the ms displays without keep (`.ID`, `.LD`, `.CD`, `.BD`).

Major quotes

Major quotes are more than one line long and need to be set apart from the rest of the text without quotation marks around them. This can be accomplished with the `.(q` and `.)q` macros. (See Table 6-17.)

Table 6-17 Major quotes macros

Type	Form	Explanation
Macro	`.(q`	Begin major quote display.
Macro	`.)q`	End major quote display.

Standard lists

Lists are indented, single-spaced, unfilled displays. They're used when the text should stand out from the normal text, as with columns of figures or examples. The macros `.(l` and `.)l` are used to display lists. (See Table 6-18.)

Table 6-18 Standard lists macros

Type	Form	Explanation
Macro	`.(l`	Begin list display.
Macro	`.)l`	End list display.

Custom lists

You can use fancier lists in me by utilizing the fill mode. By default lists are normally collected in nofill mode, but the addition of a capital F as an argument to the list macro will cause the list to be indented from both margins.

If you wish to center a list, the c argument can be used, and the L argument will left justify your list. (See Table 6-19.)

Table 6-19 Custom lists macros

Type	Form	Explanation
Macro	.(l F	Begin list in fill mode.
Macro	.)l	End list in fill mode.
Macro	.(l C	Begin centered list display.
Macro	.)l	End list display.
Macro	.(l L	Begin left-adjusted list display.
Macro	.)l	End list display.

Creating footnotes

You can produce footnotes with me with .(f and .)f. The material is collected, saved, and printed at the bottom of the current page. The footnote is printed two points smaller than the text and is separated from the main body of text by a horizontal line. (See Table 6-20.)

You can produce footnotes that are numbered automatically by placing the string ** immediately following the text to be footnoted.

Table 6-20 Begin and end footnote macros

Type	Form	Explanation
Macro	.(f	Begin footnote.
Macro	.)f	End footnote.

Creating an index or a table of contents

You should enclose all entries you want placed in an index in the me begin and end delimiters .)x and .)x.

Understanding index format

Material to be printed in an index or table of contents should be placed between the `. (x` and `.) f`. (See Table 6-21.)

Table 6-21 Index format macros

Type	Form	Explanation
Macro	`. (x` [*n*]	Begin index entry, where *n* is the page number of the entry. If *n* = `_`, no page number or line of dots will be printed. If *n* = `" "`, a line of dots will appear with no page number. If *n* is any other character, it will be understood as the name of the index, thus allowing several indexes to be run simultaneously.
Macro	`.) x`	End index entry.

Printing the index

The `. xp` macro is used in me to print a formatted list of the text items designated by the index macros. In me, the index must be printed at the end of the paper, rather than at the beginning. You will have to rearrange the paper physically after printing if you wish to use the me index as a table of contents. (See Table 6-22.)

Table 6-22 Index print macro

Type	Form	Explanation
Macro	`. xp`	Print index.

Drawing boxes

You can draw a box around a single word or a group of words with the box macros.

Boxing a word

Use the `.bx` command in me to draw a box around a single word (see Table 6-23). The word to be boxed is entered as an argument to the macro.

Table 6-23 Boxed word macro

Type	Form	Explanation
Macro	`.bx`	Draw a rectangular box around a word, where x is any word.

Boxing a block of text

You can box a phrase in me by grouping the arguments to the `.bx` command in double quotation marks, but you cannot box more than one line at a time.

Checking your work

You can check your file for formatting errors with the `checknr` program. `checknr` examines your file and reports any unrecognized macros or unbalanced macro constructions.

To run `checknr`, enter the command

`checknr` *file*

Any discrepancies are written to the standard output. Or, if you prefer, you can direct the output from `checknr` to a file so you can examine it later:

`checknr` *file* > *output-file*

For more detailed instructions on using this program, refer to `checknr`(1) in *A/UX Command Reference.*

Creating your own macros

You can create your own macro out of a sequence of `nroff/troff` commands and other defined macros. The basic procedure is to set up a definition string and then name the macro with uppercase letters. In `me`, avoid using `TS`, `TH`, `TE`, `EQ`, and `EN`.

Conventions used in this reference

The following conventions are used to describe macro names:

n   Digit

x   Alphanumeric character

All other characters are literals (characters that stand for themselves).

Macro and string names are kept in a single internal table. Therefore, there must be no duplication among such names. Number register names are kept in a separate table.

Defining a macro in `me`

To define a macro in `me`, start the definition with `.de` *XX*, where *XX* is the name you wish to call the macro. List any requests, or other macro commands that will be included, then end the macro with `..` on a separate line. Your macro will now be named *XX*.

Reference tables

Table 6-24 me macro summary

Name	Description
.b [x]	Print x in boldface. If x is not present, print all subsequent text in boldface.
.bc	Begin new column.
.bi [x]	Print x in bold italics. If x is not present, print all subsequent text in bold italics.
.bp	Begin new page.
.bx	Print inside a box.
.(b	Begin block text.
.)b	End block text.
++B	Print bibliography.
.ce [x]	Center x lines. x defaults to 1.
.(c	Begin centered block.
.)c	End centered block.
.+c T	Print chapter with title T.
.en	End equation.
.eq [x y]	Begin equation.
	$x=$ I Indented
	$x=$ L Left-adjusted
	$x=$ C Centered
	$y=$ E Equation label
.fo	Print footer.
.(f	Begin footnote.
.)f	End footnote.
.in [+n]	Indent n spaces. n can be a negative number for left indent.
.he	Print header.
.i [x]	Print x in italics. If x is not present, print all subsequent text in italics.
.ip [x y]	Indented paragraph, where x is a lable, and y is the indentation.

Table 6-24 me macro summary (*continued*)

Name	Description
.lp	Left-adjusted paragraph.
.(l	Begin list.
.)l	End list.
.np	Numbered paragraph.
.nr	Numbered register.
.pp	Standard paragraph.
.++P	Print preliminary part of paper.
.(q	Begin major quote.
.)q	End major quote.
.r [x]	Print x in roman. If x is not present, print all subsequent text in roman.
.sh	Print section heading.
.sm	Decrease point size by 2.
.sp [n]	Space down n. n defaults to 1.
.te	End table.
.th	Set up Berkeley thesis environment.
.th	End running table heading.
.ti[+n]	Temporarily indent n spaces .n can be a negative number for left indent.
.ts [H]	Begin table. H indicates multipage header.
.uh	Unnumbered section heading.
.ul [x]	Underline x.
.xp	Print index.
.(x [x y]	Begin index entry. x is the page number and y is the amount of indentation in ens.
.)x	End index entry.
.(z	Begin floating keep.
.)z	End floating keep.
.1c	Resume one-column printing.
.2c	Begin two-column printing.

Table 6-25 Number register summary

Name	Description
pp	Standard paragraph point size
	Initial value: 10
sp	Section header point size
	Initial value: 10
tp	Title page point size
	Initial value: 10

Table 6-26 String summary

Name	Description
\ * #	Delayed text

7 `tbl` Tables

This chapter explains `tbl` and how you can use it to obtain the output you desire. However, the best way to learn `tbl` is by studying the examples in "Examples of `tbl` Input and Output" and by creating your own practice exercises based on the samples provided.

What is `tbl`?

The `tbl` program is a document-formatting preprocessor for `troff` and `nroff`. Tables consist of columns that can be independently centered, right adjusted, left adjusted, or aligned by decimal points. Headings can be placed over single columns or groups of columns. A table entry can contain equations or consist of several rows of text. Horizontal or vertical lines can be drawn within the table, and any table or element can be enclosed in a box.

The `tbl` program converts table-formatting instructions into `nroff/troff` commands, and these processors do the actual formatting of the text.

Using `tbl`

You can use `tbl` commands with both `nroff` and `troff`, with the restrictions noted in the next section. Three components are required to use `tbl`: `.TS` and `.TE` commands, the data that fills the columns, and instructions on organizing the rows and columns.

Understanding command-line syntax

`tbl` can be run on a simple table with the command

`tbl` *file* | `troff`

For more complicated use, where there are several input files and macro package commands (such as mm) as well as tables, the command would be

`tbl` *file1 file2 file3* | `troff` -mm

The files are processed in sequence, and then this data is passed on to the remaining processors.

If a filename is not specified on the command line or if the filename given is a minus sign (–), `tbl` reads the standard input.

Using `tbl` with the `nroff` formatter is similar to using it with `troff`, but only certain hard-copy terminals can print vertical lines or boxed tables.

For the convenience of those using line printers without adequate driving tables or post-filters, there is a special `-TX` command-line option to `tbl` that produces output without fractional-line motions (see `tbl`(1) in *A/UX Command Reference*).

The only other command-line options recognized by `tbl` are the `ms` and the `mm` macros. These arguments are accepted by `tbl`, but it is usually more convenient to place them on the `nroff`/`troff` formatter portion of the command line.

Defining table formats

The general format of `tbl` input in a document is

preceding text
`.TS`
global options;
column formatting instructions.
table data
`.TE`
more text

The global format line defines the overall format of the table. The column formatting line defines the column alignment of the table entries. These specifications are preceded and followed by a table start (`.TS`) and table end (`.TE`) command. The `.TS` and `.TE` lines are then used by `troff` as command delimiters.

If there isn't enough space on the page for a table, it is continued on the next page; however, boxes and vertical lines aren't drawn properly if a table is split between two pages. Enclosing your table in display macros keeps your table on one page (see Chapter 4, "`mm` Macros," and Chapter 5, "`ms` Macros," for a discussion of the display macros).

Using global format options

The global format line affects the format of the whole table. It consists of a single line of instructions and must immediately follow the `.TS` command. Option names must be

separated by spaces, tabs, or commas, and the line must be terminated by a semicolon. Allowable global options are listed in Table 7-1.

Table 7-1 Allowable global options

Global option	Description
allbox	Enclose each table entry in a box.
box	Enclose the table in a box.
center	Center the entire table.
delim *xx*	Specify that characters *xx* will be used as eqn delimiters.
doublebox	Enclose the table in a double-ruled box.
expand	Expand the table to the width of the current line length.
linesize *n*	Increase line thickness (for example, box and allbox) to *n* point size.
tab(*x*)	Separate data items with character *x* instead of tab.

Global options are discussed in the following sections.

Setting table width and positioning

The default positioning for a table produced by tbl is left adjusted. The center option places the table in the center of the page. The expand option spreads the table across the full width of the current line length of the page (see Figure 7-1).

Drawing boxes

There are three ways to globally specify boxed tables: box encloses the table in a single box, allbox encloses each item of the table in a box, and doublebox encloses the table in a double-lined box. Each is illustrated in "Examples of tbl Input and Output" later in this chapter.

The tbl program tries to keep a boxed table on one page by issuing the appropriate .ne (need) troff command. This command is calculated from the number of lines in the table. If there are spacing commands embedded in the input, however, the .ne

commands may be inaccurate. In that case, you can use `troff` keep-release macros or can manually specify the `.ne` *n* command. If a multipage table is required, use the `.TS H` and `.TH` macros designed for this purpose (see "Producing Multipage Tables With Repeated Headings" later in this chapter).

Changing line thickness

The `linesize` *n* (where *n* is a point size) option permits you to specify a heavier line in your table than the default 10-point.

Setting a new tab character

`tbl` uses the tab character to separate items of data. Because tabs are invisible, it is useful to reset the tab character to some other character that can be seen. You do this with the `tab` (*x*) option, where *x* is a character you will not need in your table. For example, to change the tab character to a colon (:), use the following command:

`tab(:)`

Using mathematical equations in tables

When `tbl` processes columns of numbers, it looks for a decimal point and attempts to split numeric format items into two parts (see "Understanding Numeric Columns" later in this chapter). This feature interferes with the way `eqn` processes equations. The `delim` *xx* global option enables you to define `eqn` delimiters within your table, preventing this interference.

◆ **Note** It is still better to avoid putting equations in numeric (*n*-style) columns. ◆

Using `tbl` with other A/UX preprocessors

When `pic`, `tbl`, and `eqn` operate on the same file, `pic` is always called first:

```
pic file | tbl | eqn | troff
```

If only `eqn` and `tbl` are present, `tbl` should be called first. `eqn` produces a larger expansion of the input, and it is faster and more efficient to execute it after `tbl`. If there are no equations within tables, either sequence works. However, if there are equations within tables, `tbl` must be called first, or the output will be scrambled.

When there are several input files containing tables, equations, and mm macros, the correct command sequence is

```
tbl file1 file2 file3 | eqn | troff -mm
```

If you also use the extended mathematical character set in `/usr/pub/eqnchar` (see Chapter 8, "`eqn` Equations"), the command reads

```
tbl /usr/pub/eqnchar file | eqn | troff -mm
```

Aligning columns: Keyletters

The format line(s) specifies column layout. It contains a "keyletter" for each column of the table that represents a particular column format instruction.

Keyletter instructions may be entered in either uppercase or lowercase, and the last entry in the format section is always followed by a period. Keyletters are listed in Table 7-2.

Table 7-2 Keyletter descriptions

Keyletter	Description
a	Alphabetic column entry; entries are left-adjusted and positioned so the widest entry is centered within the column.
c	Centered column entry.
l	Left-adjusted column entry.
n	Numeric column entry; entries are aligned so the numbers line up at a decimal point.
r	Right-adjusted column entry.
s	Spanned heading; the entry from the previous column continues across this column.
^	Vertically spanned heading; the entry from the previous row continues down through this row.

Understanding numeric columns

When numeric column alignment (n-style) is specified, 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. If an alignment or alignment character isn't specified, the item is centered. However, the special nonprinting character string (\ &) can 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 Table 7-3, items shown in the "Input" column will be aligned in a numeric column as shown in the "Output" column.

Table 7-3 Numeric column alignment

Input	Output	Comments
4.2	4.2	Aligned by decimal point
13	13	No alignment character
26.4.12	26.4.12	Aligned by decimal point
749.12	749.12	Aligned by decimal point
abcdefg	abcdefg	Centered
abcdefg\&	abcdefg	\ & as alignment character

If numeric data is used in the same column with wider l- or r-type table entries, the widest number is centered relative to the widest nonnumeric item; for example,

```
.TS
center tab(:) ;
l l
n n.
shortest:longest entry
13:13
42,347.99:42,347.99
0.5:0.5
.TE
```

will send the output

shortest	longest entry
13	13
42,347.99	42,347.99
0.5	0.5

This is similar to alphabetic subcolumns (a-style), which are always slightly indented relative to left adjusted items. If necessary, the column width is increased to force this.

How tbl reads keyletter instructions

The layout of keyletters in the format section represents the layout of the actual data in the table. For example, a simple three-column format might appear as

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

The first line of this table contains a centered heading spanned across all three columns (c s s). Each remaining line contains a left-adjusted item in the first column followed by two columns of numeric data (l n n). These specifications produce the following:

	Spanned Heading	
Item-1	34.22	9.1
Item-2	12.65	.02
Item-3	23	5.8
Total	69.87	14.92

Successive line formats separated by commas can also be given on the same line. For example, the format for the preceding example could be written

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

Spaces between the keyletters are not required, but they can be helpful visually when setting up or changing a table format. Each line in the format section corresponds to a single line of data. However, if there are more lines of data than there are format lines, the last format line corresponds to all following data lines up to the table end (`.TE`) command or a table continue (`.T&`) command (see "Adding New `tbl` Format Instructions in the Text" later in this chapter).

Fine-tuning keyletter specifications

To permit further refinement of your table formatting instructions, keyletters can be followed by qualifiers that change the format and placement of the column entries, or change the size and shape of the columns.

These qualifiers can be in any order, they can be uppercase or lowercase, and they need not be separated by spaces (except as indicated). For example,

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

specifies a numeric column entry in 12-point type with a maximum width of 2.5 inches, in italic font and separated by 6 ens from the next column entry.

Drawing horizontal lines

A keyletter can be replaced by an underscore character (_) or equal sign (=) to specify a single or double horizontal line in place of the column entry:

```
l _ l
```

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. (See Figure 7-7.)

Drawing vertical lines

A vertical bar (|) may be placed between keyletters to cause a vertical line between the corresponding columns of the table (see Figure 7-1). A vertical bar to the left of the first keyletter or to the right of the last one produces a line at the edge of the table. If two vertical bars appear between keyletters, a double vertical line is drawn, for example,

```
| l || l |
```

Setting column spacing

A number may follow the keyletter to indicate the amount of separation between this column and the next column, for example,

```
n6 n
```

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 equal to half the current type size. If the `expand` option is used, these numbers are multiplied by a constant, making the table as wide as the current line length. The default column separation number is 3. If the separation is changed, the largest space commanded is assumed.

Setting vertical spacing

A keyletter followed by `v` and a number indicates the vertical line spacing within a multiline table entry. The number may be plus or minus (+ or –), in which case it is taken as an increment or decrement from the current vertical spacing, for example,

```
cv+2
```

A column separation space value must be separated by blanks or some other specification from a vertical spacing command. This command has no effect unless the corresponding table entry is a block of text (see "Setting Up Text Blocks for Multiline Entries" later in this chapter).

Setting vertical spanning

Vertically spanned items extending over several rows of the table are normally centered in their vertical range. If a keyletter is followed by t, any corresponding vertically spanned item will begin at the top line of its range, for example,

```
lt ct at
```

Setting column width

A keyletter followed by w and a value in parentheses specifies maximum column width; for example,

```
lw(2i)
```

specifies a 2-inch column.

If the largest element in the column is not as wide as the width value given after the w, the column 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 (see "Setting Up Text Blocks for Multiline Entries" later in this chapter).

Normal troff formatter units can be used to scale the width value. The default value is *ens*, but inches also may be 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.

Setting equal-width columns

A keyletter followed by e indicates equal-width columns. All columns whose keyletters are followed by e or E are made the same width, for example,

```
le ne
```

Setting staggered columns

A keyletter followed by 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** Staggered columns do not work with the allbox option. ◆

Changing fonts

A keyletter followed by f and a string containing a font name (such as R, I, or B) or font number (such as 1, 2, or 3) indicates that the corresponding column should be in a different font from the default font. For example,

```
1f2 1fB
```

specifies one column of italics and one column of boldface.

All font names are one or two letters. A one-letter font name should be separated from whatever follows by a space or tab.

◆ **Note** troff font change commands given within the table data override these specifications.◆

Changing point sizes

A keyletter followed by p and a number indicates the point size of the corresponding table entries. If the number is preceded by a plus (+) or minus (–) sign, the value is incremented or decremented from the current point size, for example,

```
1p8
```

If both a point size and a column separation value are given, one or more blanks must separate them.

Using zero-width items

A keyletter followed by a data item is 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.

Using default column spacing

Column descriptors missing from the end of a format line are assumed to be left adjusted. The longest line in the format section, however, defines the number of columns in the table. Extra columns in the data are ignored.

Refining formats

Table data is entered immediately following the format line. Each line of the table is entered as one line of data. Very long input lines can be broken up, however, by ending the first part of the input line with a backslash (\) or by using text blocks (see "Setting Up Text Blocks for Multiline Entries" later in this chapter). When using the backslash, the line following it is combined with the preceding line (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.

Inserting `troff` commands in tables

`troff` commands can be interspersed with table data to provide further refinement and definition of the table output.

An input line beginning with a dot and followed by anything but a number is assumed to be a command to `troff` and is passed through unchanged, retaining its position in the table. For example, an `.sp` command can be used within a table to change the spacing between rows.

Point size and font changes may also be made within the table data. `troff` commands (such as `\fI`, `\s+2`, and so forth) entered within the table override `tbl` column-formatting instructions.

Setting up text blocks for multiline entries

In order to include a block of text as a table entry, precede it by tab and `T{`. Enter text on a new line, and terminate it with "`T{`" –for example,

previous text^`IT{`

block of

text

`T}`

where `^I` is a tab character or other character defined as a tab character in the global specification of the table. The begin delimiter (`T{`) must be followed by a new line, and the end delimiter (`T}`) must begin a new line; however, additional columns of data may

follow after a tab on the same line. Text is pulled out from the table, processed separately by the formatter, and replaced in the table as a solid block.

◆ **Note** Limits in the `troff` program will 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 is used:

$l \mp c / (n + 1)$

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 such as point size or font used in formatting the text block are

■ those defined for your whole document (including the effect of the `.TS` macro)

■ any table format specifications of size, spacing, font, and column keyletters

■ `troff` commands within the text block itself (commands within the table data but not within the text block do not affect that block)

Drawing lines

In addition to specifying lines using the keyletter system, `tbl` also permits line specification within the data section.

Drawing full-width horizontal lines

If an input line contains only an underscore character (_) or equal sign (=) on a line by itself, a single or double line is drawn that extends the full width of the table, for example,

```
.TS
```
global options ;
column formatting instructions .
data

—

data

—

```
.TE
```

Drawing single-column-width lines

If an individual table entry contains an underscore character (_) or equal sign (=), a single or double line is drawn that extends the full width of the column. Such lines are extended to meet horizontal or vertical lines adjoining this column.

To obtain these characters (_ and =) explicitly in a column, they should be preceded by a \& or followed by a space before the usual tab or newline character.

An input table entry that contains only the string _ is assumed to be a single line as wide as the text in the column. It differs from the above single-column line in that it is not extended to meet adjoining lines.

Repeating characters

An input table entry containing only the string \Rx, where x is any character, is replaced by repetitions of that character as wide as the data in the column. This sequence of characters is not extended to meet adjoining columns.

Using vertical spanning

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 the keyletter '^'.

Producing multipage tables with repeated headings

You can print tables on more than one page with `tbl`, and if you use the `mm` and `ms` macros, you can produce multipage tables with repeated headings. Begin your table with this macro:

```
.TS H
```

After you enter your heading text, input the macro `.TH`. Text that precedes the `.TH` is placed at the top of each page of the table. The remaining lines of the table are placed on additional pages as required, for example,

```
.TS H
```
global options ;
column formatting instructions .
heading text
```
.TH
```
data
```
.TE
```

If you use the `mm` macro package, the `.TH` macro can take the argument `N`. This causes the table header to be printed only on the first line on a 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 formatting instructions .
heading text
```
.TH
```
data
```
.TE
.TS H
```
global options ;
column formatting instructions .
heading text
```
.TH N
```
data
```
.TE
```

◆ **Note** This is not a feature of tbl but of mm and can be used only with the mm macro package. ◆

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.

Adding new tbl format instructions in the text

The table continue command (.T&) resets column parameters. It is used to specify tables with groups of rows containing identical formats. Each group is different, but within a group the format is the same.

Table specifications are split into groups (separated by .T&), and each set of instructions specifies the format of each group. (See Figure 7-5.)

The .T& command is recognized only within the first 200 lines of a table and does not change global options, the number of columns, the spacing between columns, or the selection of equal-width columns.

An example of such 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 located close to its corresponding format line.

tbl restrictions

Input to tbl is subject to the following restrictions:

- The tbl program accepts up to 35 columns; the actual number that can be processed may be smaller depending on the availability of troff number registers.
- The keyletters n and a may not be used in the same column.
- Computation of column width is restricted to the first 200 lines of data.
- Table continue commands (.T&) apply to only the first 200 lines of a table.
- Staggered column entries and multipage tables do not work with the global option allbox.
- When calculating column widths, all entries are assumed to be in the font and point size in use when the .TS request was encountered. However, font and point size specifications can be changed within the data section (as in the entry \s+3*data*\s0).
- When processing a file that contains tables and equations, tbl should always be called before eqn.
- Number register names used by tbl must not be used within tables. These include two-digit numbers from 31 to 99 and strings of the form $4x$, $5x$, $\#x$, $x+$, x |, $\wedge x$, and $x-$, where x is any lowercase letter. The names ##, #-, and #^ should also be avoided. (When assigning eqn delimiters in a table, the symbols ## must never be used.)
- Multipage tables should not be boxed.
- No more than 30 text blocks can be used in a table. This number may be smaller if the individual text blocks are long.
- Table width is defined in number register TW before the .TE macro is invoked and may be used to expand that macro.

Examples of `tbl` input and output

Figures 7-1 through 7-10 are included to show `tbl` input and output information and to illustrate the basic concepts of the `tbl` program. Although each figure has a title naming certain options or features, other uses of `tbl` can be learned from them as well. For instance, Figure 7-5 shows the use of additional command lines and also specifies bold type print in the format area. Studying these examples will help you learn how to use the `tbl` program much more easily than by simply reading the written explanations.

Input:

```
.TS
expand box center tab(:) ;
c s
l | l .
Menu
_
Monday:Fish
Tuesday:Tostada
Wednesday:Tuna salad
Thursday:Spaghetti
Friday:Chicken
.TE
```

Output:

Menu	
Monday	Fish
Tuesday	Tostada
Wednesday	Tuna Salad
Thursday	Spaghetti
Friday	Chicken

Figure 7-1 Table using the `expand` option

Input:

```
.TS
allbox center tab(:) ;
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
3:47-55:2.87
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 7-2 Table using the allbox and center options

Input:

```
.TS
center box tab(:);
cB s s
cI | cI | cI |
l | l | n .
Major New York bridges
_
Bridge:Designer:Length
_
Brooklyn:J. A. Roebling:1595
Williamsburg:L. L. Buck:1600
_
::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
Triborough	O.H. Ammann	1380
		383
Bronx Whitestone	O.H. Ammann	2300
Throgs Neck	O.H. Ammann	1800

Figure 7-3 Table using the vertical bar keyletter feature

Input:

```
.TS
center 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 7-4 Table using horizontal lines in place of keyletters

Input:

```
.TS
center box tab(:) ;
cfB s s s .
Composition of foods
=
.T&
c | c s s
c | c s s
c | c | c | c .
Food:Percent by weight
\^:_
\^:Protein:Fat:Carbo-
\^:\^:\^:hydrate
_
.T&
l | n | n | n .
Halibut:18.4:5.2:...
Lima beans:7.5:.8:22.0
Mushrooms:3.5:.4:6.0
.TE
```

Output:

Composition of foods			
Food	Percent by weight		
	Protein	Fat	Carbo-hydrate
Halibut	18.4	5.2	...
Lima beans	7.5	.8	22.0
Mushrooms	3.5	.4	6.0

Figure 7-5 Table using additional command lines

Input:

```
.TS
center allbox tab(:) ;
cfI s s
clw(1i) clw(1.3i) clw(1.3i)
1 1 1 .
New York area rocks
Era:Formation:Age (years)
Precambrian:Reading:>1 billion
Paleozoic:Manhattan:400 million
Mesozoic:T{
Newark Basin, incl. Lockatong
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	>1 billion
Paleozoic	Manhattan	400 millior
Mesozoic	Newark Basin, incl. Lockatong	200 million
Cenozoic	Coastal Plain	On Long Island 30,000 years; cretacious sediments redeposited by recent glaciation

Figure 7-6 Table using text blocks

Input:

```
.TS
center delim $$ tab(:) box ;
cp12b | c | c
l | c | c .
1:$ rho $:$ sigma $
=
$ omega sub 1 $:$ i over 2 $:$ x sub i $
_
$ pi sub 2 $:$ i over -2 $:0
_
$ theta sup 1 = omega sub 3 $:$ i over 2 $:$ rho $
_
$ lambda sub 2:0 :$ x + y over 2 $
.TE
```

Output:

1	ρ	σ
ω_1	$\frac{i}{2}$	x_i
π_2	$\frac{i}{-2}$	0
$\theta^1{=}\omega_1$	$\frac{i}{2}$	ρ
λ^1_2	0	$x{+}\frac{y}{2}$

Figure 7-7 Table using eqn delimiters

Input:

```
.TS
center tab(#) delim $$ ;
c c c | c .
$P$#$Q$#$R$#$P ~ cap ~ ( wig Q ~ cup ~ R ) $
=
T#T#T#T
_
T#T#F#F
_
T#F#T#T
_
T#F#F#T
_
F#T#T#F
.TE
```

Output:

P	Q	R	$P \cap (\sim\!Q \cup R)$
T	T	T	T
T	T	F	F
T	F	T	T
T	F	F	T
F	T	T	F

Figure 7-8 Table using horizontal lines in place of data

Input:

```
.TS
center tab(:) ;
l c c c l
l c c c l
l | c | c | c | l
l c c c l
l c c c l
l c c c l
l | c | c | c | l
l c c c l
:\(da::\(da
:_::_:
:lex::yacc:
:_::_:
:\(da::\(da
:_::_:
Input \(->:yylex:\(->yyparse:\(-> Output
:_::_:
.TE
```

Output:
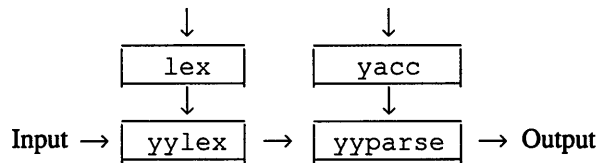
Figure 7-9 Table showing the versatility of the tbl program

Input:

```
.TS
center box tab (:) ;
cB              |           cB              |           cB
cf1             |           cf3             |           cf2 .
Roman:Bold:Italic
=
a : a : a

_
b : b : b

_
c : c : c

_
d : d : d

_
e : e : e
.TE
```

Output:

Roman	Bold	Italic
a	a	a
b	b	b
c	c	c
d	d	d
e	e	e

Figure 7-10 Table showing font changes

8 eqn Equations

This chapter shows you how to use eqn. Examples are provided to illustrate its syntax and rules of grammar. Study these examples, and in a very short time you should be able to produce typeset-quality mathematical text.

What is eqn?

The eqn program is a mathematical equation-formatting preprocessor for troff and nroff. It was designed to be easy to learn and use. Its language has few rules, and even fewer exceptions, and can be learned very quickly. It interfaces directly with troff, so mathematical expressions can be embedded in the running text of a manuscript, and the entire document can be produced in one process.

Typical mathematical expressions require point size and font changes, positioning, line drawing, and other functions to print according to mathematical conventions. In the eqn program these are done automatically; eqn converts mathematical input into troff commands, and the resulting output is passed directly to the formatter for further processing.

Using eqn

eqn needs no special keys to enter even the most complicated equations. Subscripts and superscripts are printed automatically in the appropriate size and font. Fraction bars are made the right length and positioned at the correct height. Output may be produced on either a typesetter, a laser printer, or a terminal with forward and reverse half-line motions.

Understanding command-line syntax

To produce typeset-quality mathematical text, use the following command:

eqn *file* | troff

Any troff options (such as mm) are located following the troff formatter part of the command:

eqn *file* | troff -mm

An nroff-compatible version of eqn (neqn(1)) can be used with hard-copy terminals that have half-line forward and reverse capabilities. The input language is identical, but some things will not look as good because these terminals do not provide

the same variety of characters, sizes, and fonts. However, the output is usually adequate for proofreading.

To print equations on one of these devices, use the command

```
neqn file | nroff
```

or

```
neqn file | nroff -Tx
```

where x is the terminal type being used.

Using eqn with other A/UX preprocessors

When eqn operates on the same file as the other A/UX preprocessors, tbl and pic (see Chapter 7, "tbl Tables," and Chapter 9, "pic Line Drawings"), pic should be called first:

```
pic file | tbl | eqn | troff
```

If only eqn and tbl are present, tbl precedes eqn:

```
tbl file | eqn | troff
```

eqn produces a larger expansion of output than tbl, and it is faster and more efficient to produce the table first and the equation last. The order is optional, however, unless there are equations within tables, in which case tbl must be called before eqn or the output will be unreadable.

Using Greek letters and mathematical symbols

eqn knows the Greek alphabet and most mathematical symbols and mathematical names. For example, the input

```
.EQ
size +2
{e sup {i delta t}}
.EN
```

produces the output

$e^{i\delta t}$

Braces can also occur within braces if necessary. For example, the statement

```
.EQ
size +4
{e sup {i pi sup {rho +1}}}
.EN
```

generates

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

Each string of characters (delimited by spaces, tildes, carets, or tabs) is compared with a symbol table. If `eqn` finds the string contained there, it substitutes the `troff` translation of that string. Digits, parentheses, brackets, punctuation marks, and the following mathematical words are converted to roman font:

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

Other strings are converted to italic font. In the previous example, `pi` and `rho` become their Greek equivalents (π and Γ). Parentheses, digits, and operators are also produced in roman font.

A common error is to type `f(pi)` without leaving spaces on both sides of the `pi`. Without spaces, `eqn` does not recognize `pi` as a special word, and it appears as *f(pi)* in the output instead of *f(π)*.

The only way `eqn` can deduce that some sequence of letters is 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 emphasized by surrounding them with tildes or carets. The following:

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

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

$x = 2 \pi \int \sin (w\ t)$

Tables 8-1 and 8-2 provide a complete list of the mathematical characters recognized by `eqn`.

Table 8-1 Standard mathematical characters

Input	Output
`>=`	$\geq$
`<=`	$\leq$
`==`	$\equiv$
`!=`	$\neq$
`+-`	$\pm$
`->`	$\rightarrow$
`<-`	$\rightarrow$
`<<`	$\ll$
`>>`	$\gg$
`inf`	∞
`partial`	∂
`half`	$\frac{1}{2}$
`prime`	$\prime$
`approx`	$\approx$
`nothing`	
`cdot`	$\cdot$
`times`	$\times$
`del`	Δ
`grad`	∇
`dollar`	$\$$
`...`	$\cdots$
`,...,`	$,\cdots,$
`sum`	$\sum$
`int`	$\int$
`prod`	Π
`union`	$\cup$
`inter`	$\cap$

Table 8-2 Greek alphabet

Input	Output	Input	Output
alpha	α	ALPHA	*ALPHA*
beta	β	BETA	*BETA*
gamma	γ	GAMMA	Γ
delta	δ	DELTA	Δ.
epsilon	ε	EPSILON	E
zeta	ζ	ZETA	*ZETA*
eta	η	ETA	*ETA*
theta	θ	THETA	Θ
iota	ι	IOTA	*IOTA*
kappa	κ	KAPPA	*KAPPA*
lambda	λ	LAMBDA	Λ
mu	μ	MU	*MU*
nu	ν	NU	*NU*
xi	ξ	XI	Ξ
omicron	o	OMICRON	*OMICRON*
pi	π	PI	Π
rho	ρ	RHO	*RHO*
sigma	σ	SIGMA	Σ
tau	τ	TAU	*TAU*
upsilon	υ	UPSILON	Υ
phi	φ	PHI	Φ
chi	χ	CHI	*CHI*
psi	ψ	PSI	Ψ
omega	ω	OMEGA	Ω

As shown in Table 8-2, several uppercase Greek letters are not provided in the eqn package. These uppercase Greek letters may be produced using troff codes. See the reference tables in Chapter 3, "nroff/troff Formatters."

Using additional symbols

Four-character `troff` names can also be used to specify any characters `eqn` does not recognize, for example, \ (p1 for the + sign and \ (mi for the - sign. (See Chapter 3, "nroff/troff Formatters," for a complete list of `troff` character codes.)

Additionally, the file /usr/pub/eqnchar contains `nroff` and `troff` definitions of several more mathematical symbols. (See Table 8-3.) These definitions must be enclosed within `eqn` delimiters in order to be processed correctly.

Table 8-3 Additional character set

Input	Output	Input	Output
circle	o	3dot	⦂
ciplus	⊕	incl	⊑
citimes	⊗	langle	⟨
3quarter	³/₄	rangle	⟩
quarter	¹/₄	member	∈
<->	↔	nomen	∉
<=>	⟺	oppA	∀
=del	=	oppE	∃
hbar	ℏ	cup	∪
ppd	¶	cap	∩
prop	∝	subset	⊂
ang	^	!subset	⊆
angstrom	Å	supset	⊃
square	□	!supset	⊇
blot	■	bigstar	✳
bullet	•	star	⋆
empty	∅	degree	o
thf	∴	wig	~
-wig	≈	=wig	≅
>wig	≳	<wig	≲

For users who are experienced with troff motion commands and string definitions, almost any mathematical character can be defined. Studying the definitions contained in usr/pub/eqnchar will give you a good idea of how this is done (see eqnchar(5) in *A/UX Programmer's Reference*).

◆ **Note** When you are making your own character definitions, it is easier if you use a line gauge from a graphics supply store to gauge the appropriate size changes and vertical and horizontal troff motions.

Using /usr/pub/eqnchar

To process a document containing the extended mathematical set (/usr/pub/eqnchar), you must include this file in your command:

eqn /usr/pub/eqnchar *file* | troff

Or, if you have also included tables in your text, you must include this file:

tbl /usr/pub/eqnchar *file* | eqn | troff

You may substitute neqn and/or nroff in both of these commands if your output device requires it.

Using command delimiters

Mathematical expressions are entered by beginning and ending each equation with the delimiters .EQ and .EN as follows:

.EQ
equation-specifications
.EN

Using displayed equations

A displayed equation is printed as a block, preceded and followed by half a vertical space (one blank line). It is specified with the mm display macro

```
.DS
.EQ
```
equation-specifications
```
.EN
.DE
```

By default, a displayed equation using mm is left-adjusted. However, placement options (centered, indented, or right) that override these defaults are provided. (See Chapter 4, "mm Macros," for a full discussion of the display macros.)

For example, when using mm, the input

```
.DS I
.EQ
x = f ( y over 2 ) + y over 2
.EN
.DE
```

produces an indented equation

$$x = f(\frac{y}{2}) + \frac{y}{2}$$

A centered equation can be produced with the following input:

```
.DS C
.EQ
x sub i = y sub i
.EN
.DE
```

The resulting equation will be centered on the page:

$$x_i = y_i$$

If you are not using a macro package to format your document, you can still manipulate the placement of equations within text. To obtain a centered equation in a document without using ms or mm, enter the following:

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

Using inline equations

An inline equation is printed within the text of your document. Like a displayed equation, it must be enclosed in delimiters, but instead of the `.EQ/.EN` sequence, you define a character to be the delimiter.

The most common character chosen to delimit inline equations is the dollar sign (`$`), which is defined at the beginning of the text file by entering the following:

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

These characters are then recognized by `eqn` in the subsequent text as delimiters and any text between them will be treated as an equation. For example, the input

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

would send this as output:

This is an example of an inline
equation $x + y = z$ using delimiters.

Producing something like `\-ray` is easy using the inline equation

```
$ gamma $-ray
```

`eqn` will try to keep the text between the delimiters on one line, but if the equation is very long, `troff` will break it based on the spacing of characters, not mathematical logic. This can produce awkward and inaccurate spacing, but you can prevent this by dividing the inline equation into sections:

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

To turn off the delimiters so the selected character can be used as text, enter the following into your file:

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

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

The following should be observed when using the inline equation format:

- Do not use braces, tildes, carets, or double quotation marks as delimiters, as these have special significance to the `.EQ` and `.EN` macros.

- `troff` font changes must be closed before inline equations are encountered.

Defining equations

The `eqn` definition facility permits a user to define an equation or part of an equation: define *name'...'*

Henceforth, any occurrence of *name* within an `eqn` expression will be expanded into whatever is inside the quotation marks.

Keywords like `sup`, `sub`, or `over`, or any `eqn` construction, may be included in a definition. For example, if the sequence

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

appears repeatedly throughout a document, you can save typing time by defining it:

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

This definition makes `xy` a shorthand for whatever characters occur between the single quotation marks in the definition. (Any character can be used instead of the quotation mark to mark the beginning and end of the definition, as long as it does not appear inside the definition.) After defining `xy`, the input

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

produces

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

Although definitions can use previous definitions, as in

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

an item cannot be defined in terms of itself, for instance,

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

Since x is now defined in terms of itself, problems will result. However, if this expression is used, the quotation marks protect the second x:

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

eqn keywords can be redefined with define. For example, you can specify "/" to mean over with the following statement:

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

Symbols can be defined differently in neqn and eqn with the operators ndefine and tdefine. A definition made with ndefine takes effect only when running neqn; when tdefine is used, the definition applies only to eqn. (Names defined with the define facility apply to both eqn and neqn.)

Specifying equations

An equation is specified by numeric items and mathematical operators. Each of these components must be separated from the others according to specific item separation conventions. For example, in the expression

$$\sqrt{5^2}$$

which was produced with the notation

```
sqrt 5 sup 2
```

sqrt and sup serve as operators, and the spaces between these keywords and the arguments are item separators.

How spaces are interpreted during input

Spaces and newline characters are used by `eqn` to separate pieces of input; they do not create space in the output. For example, the input

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

produces the output

$x=y+z+1$

Each distinct entity within `eqn` must be delimited by blank spaces. If items are not separated properly, `eqn` will interpret the expression incorrectly.

Using special characters to force output spacing

Varying amounts of blank space can be forced into the output by several characters:

- A tilde (~) gives a space equal to the normal word spacing in text

- A caret (^) gives a half-space.

- A tab character spaces to the next tab stop (tab stops must be set by `troff` commands).

Tildes, carets, and tabs also serve to delimit pieces of input. In these cases, blank spaces are optional. For example, the input

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

produces the output

$x = y + z$

Using quotation marks

Enclosing a string of characters in double quotation marks (" . . . ") prevents eqn from interpreting any special meaning the string might ordinarily have.

For example, to produce the expression

$$sqrt\frac{25}{\pi}$$

enter the following:

```
.EQ
"sqrt" 25 over pi
.EN
```

Omitting the quotation marks results in

$$\frac{\sqrt{25}}{\pi}$$

Quotation marks are used to force the printing of braces and certain eqn keywords that wouldn't normally be printed. For example, the input

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

prints

{ alpha is the name for α}

The " " construction is often used as a placeholder (or null item) when eqn requires something to satisfy its rules of grammar but when nothing is actually wanted in the output. For instance, eqn does not accept unmatched brackets, braces, or parentheses. However, the input

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

permits you to obtain only a right brace:

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

Combining items with braces

Braces (`{ }`) are used to keep multiple objects together in unambiguous groups. `eqn` interprets the items within a set of braces before applying the next mathematical function.

The end of a subscript or superscript is marked by a space, tilde, caret, or tab. If the subscript or superscript specification requires spaces within it, braces are used to mark the beginning and end. For example, the input

```
.EQ
size +2
{e sup {i delta t}}
.EN
```

produces the output

$e^{i\delta t}$

Braces can also occur within braces if necessary. For example, the statement

```
.EQ
size +4
{e sup {i pi sup {rho +1}}}
.EN
```

generates

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

A general rule is that a complicated string enclosed in braces can be used in place of a single character (such as x). The `eqn` program administers the appropriate formatting commands. In all cases, complete pairs of braces must be used (unless the null item specification is employed). Omitting one or adding an extra one produces an error.

Using equation labels

An equation label is specified as an argument to the equation start delimiter (`.EQ`):

```
.EQ 1.5c
a + b + c over abc = sqrt 25
.EN
```

The equation label is printed in the right margin:

$$a+b+\frac{c}{abc} = \sqrt{25}$$

Entering equations

The eqn program uses operators to adjust the equations as you specify, thus creating subscripts and superscripts, fractions, square roots, and diacritical marks, for example.

Subscripts and superscripts

Subscripts and superscripts are specified with the operators sub and sup. The words sub and sup must be surrounded by spaces. For example, specification

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

produces the following expression:

$$x^2 + y_k$$

The eqn program makes the necessary point size changes and vertical motion adjustments and automatically returns to the original base line. Either a space or tilde marks the end of a subscript or superscript.

Multiple levels of subscripts or superscripts are permitted, such as subscripted subscripts and superscripted superscripts. If the subscript follows the superscript, the items are grouped to the right, as in the expression

$$x^{y_z}$$

produced with the input

```
.EQ
size +2
{x sup y sub z}
.EN
```

However, if the subscript precedes the superscript

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

the items are printed one above the other.

$$x_z^y$$

Fractions

Fractions are specified with the operator `over`. For example, the input

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

produces

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

The division line is positioned and made the correct length automatically.

When both a fraction and a superscript are in the same expression, `eqn` produces the superscript first. For example, the specification

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

produces

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

Square roots

The square root symbol is produced by the operator `sqrt`. For example, the input

```
.EQ
sqrt 25
.EN
```

draws the simple expression

$$\sqrt{25}$$

With the more complicated

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

`eqn` produces

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

Items with limits

Summations, integrals, and similar constructions are specified with the operators `from` and `to`.

Either `from` or `to` can be omitted, but if both are present, they must occur in that order. For example, the input

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

produces

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

The second item (`i = inf`) is enclosed in braces because it contains spaces. Braces are not necessary for the lower part (`i=0`), however, because it contains no spaces.

Other useful keywords that can replace the `sum` in the above example are

```
int          min
inter        prod
lim          union
max
```

Because characters before the `from` can be anything, the `from-to` construction can often be used in unexpected ways. The input

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

produces the output

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

Diacritical marks

Diacritical marks are produced with the following keywords:

```
x dot      ẋ
x dotdot   ẍ
x hat      x̂
x tilde    x̃
x vec      x⃗
x dyad     x⃡
x bar      x̄
x under    x̲
```

An example of an expression using diacritical marks is

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

which will send as output

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

Oversized brackets

To produce large brackets, braces, parentheses, vertical bars, floors, and ceilings that surround information that spans more than one line, use the keywords `left` and `right`:

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

This produces

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

and the input

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

produces

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

The resulting brackets are made large enough to cover whatever they enclose.

A `right` keyword cannot exist without a corresponding `left`. If the expression requires that the left be omitted, use the paired double quotation mark null construction:

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

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

Piling objects

Large braces, brackets, parentheses, and vertical bars are often used with another facility that makes vertical piles of objects. It is specified by the operator `pile`. 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 components; braces must surround the entire list. Elements of a pile can be as complicated as needed, even containing nested piles.

Three other forms of `pile` exist:

- `lpile` makes a left-adjusted pile.

- `rpile` makes a right-adjusted 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

$$sign(x) \equiv \left\{ \begin{array}{l} 1 \text{ if } x{>}0 \\ 0 \text{ if } x{=}0 \\ \text{-}1 \text{ if } x{<}0 \end{array} \right.$$

enter

```
.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-adjusted pile. The `lpile` specifications left-adjust the remaining components.

Matrixes

Matrixes are produced easily with `eqn`. For example, to specify an array such as

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

the following statement is entered:

```
.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 as they are for a pile: each element is separated by the word *above*. The `lcol` or `rcol` keyword also can be used to left-adjust or right-adjust 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 what spacing to use.

Each column must have the same number of elements. To force each column to have the same number of elements, use the keyword `nothing`, which will give the construction the proper number of elements:

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

produces

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

Aligning equations

You can align a series of equations at some vertical position (such as an equal sign) with the operators `mark` and `lineup`.

The word `mark` can appear only once in an equation. This designates the horizontal position for all subsequent input containing the keyword `lineup`. Any number of equations may be lined up following a single occurrence of `mark`. The place where `lineup` appears is aligned with the position of the previous `mark`. For example, the input

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

produces

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

`mark` does not look ahead and anticipate the requirements of the subsequent `lineup`:

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

This specification will not work because there isn't enough room for the x+y part after the `mark` remembers where the x is. In order to correctly align the equations, the following input is necessary:

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

This produces

$x=1$

$x+y=z$

◆ **Note** The `mark` and `lineup` operations do not work with centered equations. ◆

Controlling local motions

Although the `eqn` formatter tries to position things correctly on the paper, it occasionally needs fine-tuning.

The operators `back` *n* and `fwd` *n* are used to make small horizontal moves, where *n* is how far to move in hundreths of an **em** (about the width of the letter "m"). For example, `back 50` moves output back about half the width of an "m."

Similarly, output can be moved up or down with the `up` *n* and `down` *n* operators.

Changing the size and shape of fonts

By default, equations are set in 10-point type with standard mathematical font conventions, but there are times when default assumptions are not desired. Thus, point size and font change commands are provided.

Making local changes

Local point size changes are made with `size` *n,* and local font changes are made with the `roman`, `italic`, `bold`, and `fat` operators. These changes affect only the string that immediately follows; then font or point size reverts automatically to its previous settings. For example, the input

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

produces

x y

Braces are used if something more complicated than a single character is to be affected. The input

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

produces

xy z

If fonts other than roman, italic, and bold are desired, use the `font` *x* statement (where *x* is a one-character `troff` font name or number).

◆ **Note** Since `eqn` is programmed 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 point size numbers that 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:

```
size +2
```

This makes the size two points larger. (See the example in "Combining Items With Braces" earlier in this chapter.)

Making global changes

If an entire document is to be in a nonstandard point size or font, it is a nuisance to write out a point size and font change for each equation. Accordingly, you can globally set point size or font changes, which thereafter affect 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
```

Any of the `troff` font names may be used in place of `R`. The value of `gsize` can also be made a relative change with + or –.

Generally, `gsize` and `gfont` appear at the beginning of a document, but they can also appear within a document and may be changed as often as needed.

For example, in a footnote in which the size of an equation 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.

Understanding precedence rules

Each `eqn` operator is associated with a precedence; operations with higher precedence are performed before those with a lower precedence. For example, a superscript is defined as having a higher precedence than a fraction:

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

In the following example, the `eqn` operators are listed in order of increasing precedence. Operators on the same line have equal precedence.

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

If an expression contains operators of equal precedence, the order in which these operators associate decides which operation is performed first. If the operators associate to the left, the leftmost operation precedes the rightmost operation. For example, `sqrt` and `over` have equal precedence. In the expression

$$\frac{\sqrt{25}}{\pi}$$

the `sqrt` is performed before the `over`.

The following operations associate to the left:

```
over   sqrt   left   right
```

All others group to the right.

You can force a particular analysis by placing braces around expressions; for example,

```
.EQ
x sub 2 over y sub 3 + z sub 4
.EN
```

produces

$$\frac{x_2}{y_3}+z_4$$

Changing the precedence with braces

```
.EQ
x sub 2 over {y sub 3 + z sub 4}
.EN
```

results in a different equation:

$$\frac{x_2}{y_3+z_4}$$

Troubleshooting

You can detect missing delimiters and other equation errors with program aids. Using the troubleshooting devices described here should be considered the initial step in formatting a document.

Error conditions

An internal buffer in the `troff` formatter limits the size of inline equations. If a `word overflow` message is received, the limit has been exceeded. One solution is to break the equation into smaller units with the inline delimiters. Printing the equation in a display can also solve the problem. The "line overflow" message indicates that an even larger buffer has been exceeded. In this case, the equation must be broken into two separate pieces, marking each with `.EQ`/`.EN` delimiters.

◆ **Note** eqn does not warn you about equations that are too long for one line. ◆

If a mistake is made in an equation, such as omitting a brace, having one too many braces, or having an operator with a missing argument, eqn 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 you have omitted a quotation mark or you run eqn on a nonexistent file. To check a document before printing, use the command

```
eqn files > /dev/null
```

This discards the output but prints the appropriate messages on your terminal screen.
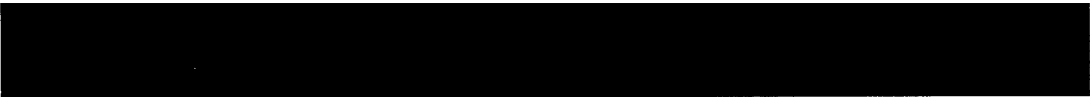
The checkeq program

The checkeq program checks for misplaced or missing delimiters. You run it with the following command:

```
checkeq file
```

Output from checkeq is written to the standard output or can be redirected to a file as follows:

```
checkeq file > output file
```

9 pic Line Drawings

This chapter explains how to use the pic preprocessor to produce simple line drawings.

What is `pic`?

`pic` is a language for including pictures and diagrams in documents produced with `troff`. It is usually used to draw relatively simple pictures, but the language can be used to describe even very complicated graphics objects.

Using `pic`

`pic` operates as a `troff` preprocessor, in the same style as `eqn` and `tbl`. Pictures are marked in the text by enclosing their descriptions between `.PS` and `.PE` pairs. The `pic` preprocessor translates these descriptions into the language understood by `troff`.

Understanding `pic` command syntax

`pic` is usually run with the command line

pic *file* | troff -mm

If equations and tables also are present, you should run `pic` before `eqn` and `tbl`:

pic *file* | tbl | eqn | troff -mm

Understanding the `troff` interface

Within `pic` specifications (`.PS` and `.PE` pairs), an input line that begins with a period is assumed to be a `troff` command and is copied to the output for further processing. Point size and font changes can be made within a `pic` specification:

```
.PS
.ps 24
circle radius .4i at 0,0
.ps 12
circle radius .2i at 0,0
.ps 8
```

```
circle radius .1i at 0,0
.ps 6
circle radius .05i at 0,0
.ps 10
.PE
```

This produces the diagram
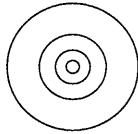
But trying to add blank lines or changing the vertical spacing within a picture interferes with the way `pic` draws objects.

Point sizes, fonts, and local motions can be manipulated within quoted strings (`"..."`) provided that whatever changes are made are reversed before exiting from the string. For example, to print text in italic font, point size 12, use

```
ellipse "\s12\f2Hello!\f1\s0"
```

This produces

Defining the picture format

A picture specification begins with a picture start command (`.PS`) and concludes with a picture end command (`.PE`). The `.PS` and `.PE` are used by `troff` as command delimiters. The general format of `pic` input is

`.PS` *optional-width*
picture-specifications
`.PE`

If *optional-width* is present, the picture is made that many inches wide, regardless of any dimensions used internally. The height is scaled in the same proportion.

If the .PS line is written

.PS < *file*

the contents of *file* are inserted in place of the picture start command (whether or not the file contains .PS or .PE).

pic copies the .PS and .PE lines from input to output intact, except that it adds two arguments to .PS:

.PS *h w*

h and *w* are the picture height and width in units.

The definitions of the .PS and .PE macros do not automatically center pictures. However, if you include the following troff instructions at the beginning of your document, your picture will be centered and offset from surrounding text.

```
.de PS
.if t .sp .3
.in (\\n(.lu-\\$2u)/2u
.ne \\$1u
..
.de PE
.in
.if t .sp .6
..
```

If .PF is used instead of the picture end command (.PE), the position after printing the picture is restored to what it was before the picture started (F is for "flyback"). Text can be overprinted on pictures, or several pictures can be superimposed.

Specifications must be separated by newlines or semicolons; a long element may be continued by ending the line with a backslash (\). Comments are introduced by a # and terminated by a newline.

If an error is made in the picture specification, `pic` generates an error message. For example, the invalid input

```
box arrow box
```

will print the message

```
pic: syntax error near line 5, file -
```

```
 context is
```

```
        box arrow ^ box
```

The caret (^) marks the place where the error is encountered; it typically follows the word in error.

Drawing pictures

Using primitive objects in `pic`, you can draw simple as well as complex pictures, change their sizes, move them in a variety of ways, add text, and group them. This section tells you how.

Drawing primitive objects

The primitive objects provided by `pic` are boxes, lines, arrows, circles, ellipses, arcs, splines (arbitrary smooth curves), and text. Most of these are shown graphically in their default sizes in Figure 9-1 (splines are shown later in this chapter).
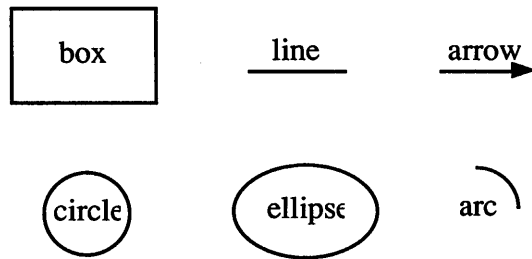
Figure 9-1 pic primitive objects

A move (see "Setting Object Attributes" later in this chapter) also is considered an object; it goes from one point to another without drawing anything, so it is an invisible object. The following keywords specify primitive objects:

```
arc        circle     move
arrow      ellipse    spline
box        line
```

The specification

```
.PS
box "BOX"
.PE
```

produces this simple box:
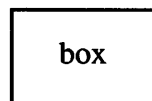
Boxes and lines may be dotted or dashed:
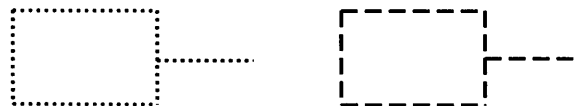
This picture was produced by

```
box dotted; line dotted; move; box dashed;\
line dashed
```

If there is a number after dotted, the dots will be that far apart. You can also control the size of the dashes. If there is a length after the word dashed, the dashes will be that long, and the intervening spaces will be as close as possible to that size. So, for instance,
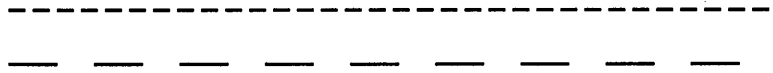
comes from this specification:

```
line right 3i dashed
line right 3i dashed 0.25i
```

Circles and arcs cannot be dotted or dashed.

A spline is a smooth curve guided by a set of straight lines; it begins and ends at the same place relative to the straight lines and in between is tangent to the midpoint of each guiding line. The syntax for a spline is identical to a line drawn along a path (see "Grouping Objects" later in this chapter):

```
spline right 1i then down .5i left 1i\
then right 1i
```

produces

Setting object attributes

Attributes describe the positioning, size, and orientation of the object. When set, they operate on a single occurrence of an object. The attributes associated with each primitive object are shown in Table 9-1.

Table 9-1 Primitive object attributes

Object	Attribute
arc	up, down, left, right, height, width, from, to, at, radius, invis, same, cw, <-, ->, <->, *text*
box	height, width, at, dotted, dashed, invis, same, *text*
circle/ellipse	radius, diameter, height, width, at, invis, same, *text*
line/arrow	up, down, left, right, height, width, from, to, by, then, dotted, dashed, invis, same, <-, ->, <->, *text*
move	up, down, left, right, to, by, same, *text*
spline	up, down, left, right, height, width, from, to, by, then, invis, same, <-, ->, <->, *text*

The keyword `at` places the geometrical center of an object in a specified place.

An object can be made invisible with the keyword `invisible` (or `invis`). This is particularly useful for positioning objects correctly near text.

For lines, splines, and arcs, `height` and `width` refer to arrowhead size. The width of an arrowhead is the distance across its tail; the height is the distance along the shaft. The arrowheads in this picture are default size:
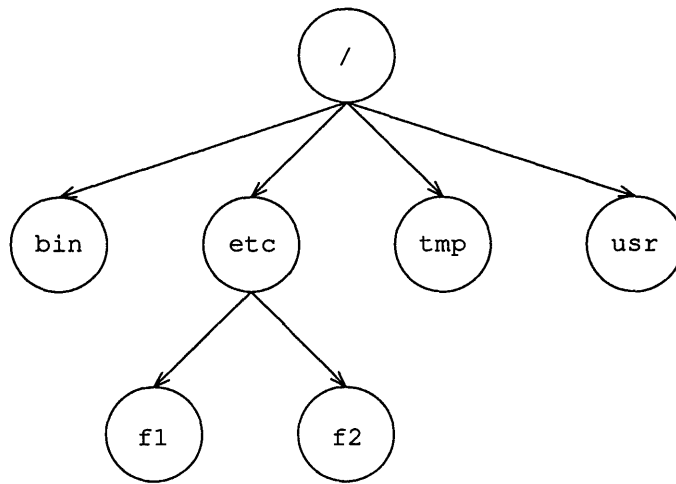
This was produced with the following code:

```
.PS
box invis height 3i wid 4i
A: circle at 0,1 "\f7/\f1"
B: circle at -1.5,0 "\f7bin\f1"
C: circle at -0.5,0 "\f7etc\f1"
D: circle at 0.5,0 "\f7tmp\f1"
E: circle at 1.5,0 "\f7usr\f1"
F: circle at -1,-1 "\f7f1\f1"
G: circle at 0,-1 "\f7f2\f1"
arrow from A.s to B.n
arrow from A.s to C.n
arrow from A.s to D.n
arrow from A.s to E.n
arrow from C.s to F.n
arrow from C.s to G.n
.PE
```

   See "Using Blocks" later in this chapter for an explanation of the letters capitalized in this code.

Dimensions are divided by `scale` during output. `pic` works internally in what it thinks are inches. Setting the variable `scale` to some value causes all dimensions to be scaled down by that value; for example,

```
scale = 2.54
```

causes dimensions to be interpreted as centimeters.

Setting object variables

A variable consists of a keyword, which may or may not be followed by a value. Keywords are used to redefine object dimensions globally.

Missing variables and values are filled in from defaults. Not all variables apply to all primitives; those that don't are ignored. Primitive object variables are listed in Table 9-2.

Table 9-2 Primitive object variables

Object	Variable	Default	Description
arc	arcrad	0.25 in.	Arc radius
arrowhead	arrowht	0.10 in.	Arrowhead height
	arrowwid	0.05 in.	Arrowhead width
	arrowhead	2.00 in.	Arrowhead style (filled)
box	boxwid	0.75 in.	Box width
	boxht	0.50 in.	Box height
circle	circlerad	0.25 in.	Circle radius
dash	dashwid	0.10 in.	Width of dashes or dots
ellipse	ellipsewid	0.75 in.	Ellipse width
	ellipseht	0.50 in.	Ellipse height
line or	linewid	0.50 in.	Line or arrow width
arrow	lineht	0.50 in.	Line or arrow height
move	movewid	0.50 in.	Width of horizontal move
	moveht	0.50 in.	Height of vertical move

These may be changed at any time, and the new values will remain in force until changed again (see the next section, "Changing the Sizes of Objects").

Changing the sizes of objects

Figures are normally drawn at a fixed scale with objects of a standard size. It is possible, however, to expand a figure to fit a particular width. If the .PS line contains a number, the drawing is forced to be that many inches wide, with the height scaled proportionately. For example,

```
.PS 3.5i
```

causes the picture to be 3.5 inches wide.

The number given as a width in the .PS line overrides the dimensions given in the picture; this can be used to force a picture to a particular size even when coordinates have been given in inches. Experience indicates that the easiest way to get a picture of the right size is to enter its dimensions in inches, then if necessary add a width to the .PS line.

You can make any object any size you want. For example, using object attributes for width and height, the input

```
box width 3i height 0.1i
```

draws a long, flat box 3 inches wide and 1/10 inch high:
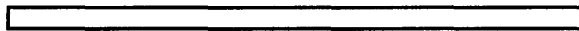
◆ **Note** This specification changes the width and height of only that particular occurrence of the object. ◆

All positions and dimensions are assumed to be in inches; specifying the "i" is optional. However, if the "i" is present, there should be no spaces between it and the number it follows.

The default size of an object can be changed by assigning values to the object variables. So if you want all your boxes to be long and skinny, and relatively close together, enter

```
boxwid = 0.1i; boxht = 1i
movewid = 0.2i
box; move; box; move; box
```

This produces
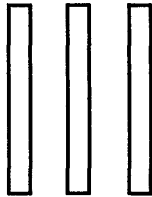
In all cases, unless an explicit dimension for some object is specified, you will get the default size. If you want an object to have the same size as the previous one of that kind, use the keyword `same`. In the set of boxes produced by the specification

```
down; box ht 0.2i wid 1.5i; move down 0.15i;
box same; move same; box same
```
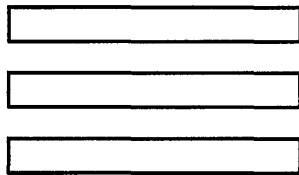
the dimensions set by the first `box` are used several times. Similarly, the amount of motion for the second `move` is the same as for the first one.
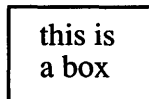
Adding text to pictures

Text is normally an attribute of some primitive; by default it is placed at the geometric center of an object. Each line of text is entered as a separate quoted string. Quotation marks are mandatory, even if the text contains no blanks. Each line is printed in the current point size and font, centered horizontally, and separated vertically by the current `troff` line spacing value.

If there are multiple text items for some primitive, they are centered vertically except as qualified. Positioning requests apply to each item independently; for example,

```
.PS
box "this is" "a box"
.PE
```

creates a standard box and centers the two pieces of text in it:

```
this is
a box
```

Text items can contain `troff` commands for size and font changes, local motions, and so on, but make sure that these are balanced so that the entering state is restored before exiting from the string.

A text item is a quoted string optionally followed by a positioning request:

"*text*" `center`
"*text*" `ljust`
"*text*" `rjust`
"*text*" `above`
"*text*" `below`

The attribute `ljust` positions the left end at the specified point, and `rjust` positions the right end at that position. `above` and `below` center the text one half-line space in the given direction.

Text is most often an attribute of some other object, but self-standing text can also be specified:

```
"this is some text" at 1,2 ljust
```

Text is centered on lines and arrows; if there is more than one line of text, the lines are centered above and below:
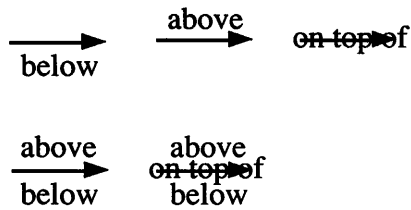
These were produced with the following specifications:

```
.PS
arrow "below" below; move
arrow "above" above; move
arrow "on top of"; move
arrow "above" "below"; move
arrow "above" "on top of" "below"
.PE
```

Positioning objects

A position is ultimately an x,y coordinate pair, and you may specify a position in this way. But a position can be specified in other ways: you may position an object in relation to a part of some other object with the move and move to commands, or by using a label embedded in a grouped object. These are discussed in the next section, "Using Coordinates."

Using coordinates

pic uses a standard Cartesian coordinate system, so any point or object has an x and a y position. The first object is placed with its start at position 0,0 by default. The x,y position of a box, circle, or ellipse is its geometrical center; the position of a line or motion is its beginning; the position of an arc is the center of the corresponding circle.

Position modifiers such as from, to, by, and at are followed by an x,y pair and can be attached to boxes, circles, lines, motions, and so forth, to specify or modify a position; for example,
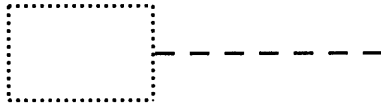
is produced by the input

```
.PS
box dotted
line dashed to 2,0
.PE
```

You can also use up, down, right, and left with line and move

```
.PS
box ht 0.2 wid 0.2 at 0,0 "1"
move to 0.5,0
box "2" same
move same
box "3" same
.PE
```

to draw three boxes, like this:
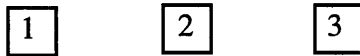
Note the use of same to repeat the previous dimensions instead of reverting to the default values.

Attributes such as ht and wid and positions like at can be written in any order:

```
box ht 0.2 wid 0.2 at 0,0
box at 0,0 wid 0.2 ht 0.2
box ht 0.2 at 0,0 wid 0.2
```

These are equivalent, although the last is harder to read and therefore less desirable.

The `from` and `to` attributes are particularly useful with arcs, to specify the endpoints. For example, these arcs
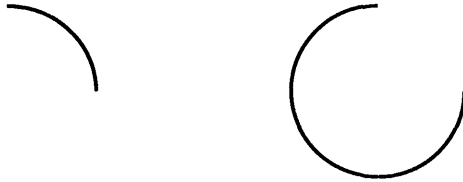
were produced with the following specifications (respectively):

```
arc from 0.5i,0 to 0,0.5i
arc from 0,0.5i to 0.5i,0
```

If the `from` attribute is omitted, the arc starts at the current position and goes to the point indicated by `to`. The radius can be made large to provide flat arcs:

```
arc -> cw from 0,0 to 2i,0 rad 15i
```

This produces
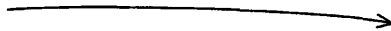
Notice that to put an arrowhead on an arc, you can use <-, ->, or <-> as an attribute.

Using corners

To cut down the need for explicit coordinates, most objects have "corners" named by compass points:
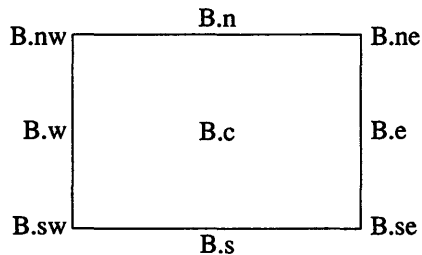
The primary compass points may also be written as `.r`, `.b`, `.l`, and `.t`, for `right`, `bottom`, `left`, and `top`. The previous box was produced with these specifications:

```
.PS 1.5
B: box "B.c"
" B.e" at B.e ljust
" B.ne" at B.ne ljust
" B.se" at B.se ljust
"B.s" at B.s below
"B.n" at B.n above
"B.sw" at B.sw rjust
"B.w" at B.w rjust
"B.nw" at B.nw rjust
.PE
```

Note the use of `ljust`, `rjust`, `above`, and `below` to alter the default positioning of text, and of a blank with some strings to help space them away from a vertical line.

Lines and arrows have a start, an end, and a center in addition to corners. (Arcs have only a start, an end, and a center.) There are many ways to indicate the corners of an object. Besides the compass points, almost any sensible combination of `left`, `right`, `top`, `bottom`, `upper`, and `lower` will work. Furthermore, if you don't like the "." notation, as in

```
last box.ne
```

you can instead say

```
upper right of last box
```

It is sometimes easiest to position objects by positioning some part of one at some part of another, for example, the northwest corner of one at the southeast corner of another. The `with` attribute in `pic` permits this kind of positioning; for example,

```
box ht 0.75i wid 0.75i
box ht 0.5i wid 0.5i with .sw at last box.se
```

produces
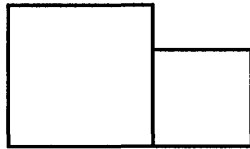
Notice that the corner after `with` is written `.sw`.

As another example, consider

```
ellipse; ellipse with .nw at last ellipse.se
```

which produces
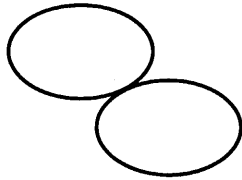
Positioning with `move`

If you want to leave a space at some designated place, use `move`:

```
box; move; box; move; box
```

This produces
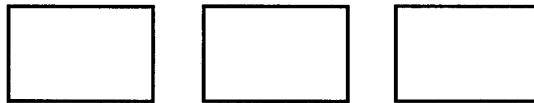
Positioning with variables

It's generally a bad idea to write everything in absolute coordinates if you are likely to change things. `pic` variables let you set parameters for your picture:

```
a = 0.5; b = 1
box wid a ht b
ellipse wid a/2 ht 1.5*b
move to Box1 - (a/2, b/2)
```

Expressions use the standard operators +, -, *, /, and %; `pic` uses parentheses, `()`, for grouping.

Probably the most important variables are those that are predefined for controlling the default sizes of objects. These may be set at any time in any picture and retain their values until reset.

You can use the height, width, radius, and x and y coordinates of any object or corner in an expression:

`Box1.x`	The x coordinate of Box1
`Box1.ne.y`	The y coordinate of the Northeast corner of Box1
`Box1.wid`	The width of Box1
`Box1.ht`	The height of Box1
`2nd last circle.rad`	The radius of the second-last circle

Any pair of expressions enclosed in parentheses defines a position; furthermore, such positions can be added or subtracted to yield new positions. If (p_1, p_2) are positions, then (p_1.x , p_2.y) refers to the point.

Labeling objects

Objects can be labeled or named for future reference, for example,

```
.PS
Box1:
        box
        #  ... other stuff...
.PE
```

Place names have to begin with uppercase letters to distinguish them from variables, which begin with lowercase letters. The name refers to the "center" of the object, which is the geometric center for most objects. For lines and motions, it refers to the beginning point.

Other combinations also work:

```
line from Box1 to Box2
move to Box1 up 0.1 right 0.2
move to Box1 + 0.2,0.1
line to Box1 - 0.5,0
```

The reserved name `Here` may be used to record the current position of some object, for example,

```
Box1: Here
```

Labels are variables; they can be reset several times in a single picture, so a line of the form

```
Box1: Box1 + 1i,1i
```

is perfectly legal.

You can also refer to previously drawn objects of each type, using the word `last`. For example, given the input

```
box "A"; circle "B"; box "C"
```

last box refers to box C, `last circle` refers to circle B, and `2nd last box` refers to box A. Numbering of objects can also be done from the beginning, so boxes A and C are `1st box` and `2nd box`, respectively.

Grouping objects

Objects are connected in the direction specified by the most recent `up`, `down`, `left`, or `right` (either alone or as part of some object), with the entry point of the second object attached to the exit point of the first. For example,

```
arrow left; box; arrow; circle; arrow
```

produces
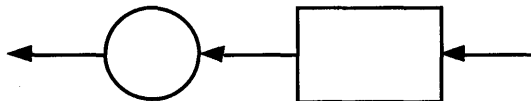
`left` indicates connection toward the left. This could also be written as

```
left; arrow; box; arrow; circle; arrow
```

Entry and exit points for boxes, circles, and ellipses are on opposite sides and at the start and end of lines, motions, and arcs.

By default, arcs are drawn 90 degrees counterclockwise from the current position. To change the direction to clockwise, use this command:

```
arc cw
```

For example, the specification

```
line; arc; arc cw; arrow
```

produces

Lines and arrows are easily drawn by specifying amount of motion and direction. Accordingly, the words up, down, `left`, and `right` and an optional distance can be attached to `line`, `arrow`, and move. For example,

```
.PS
line up 1i right 2i
arrow left 2i
move left 0.1i
line <-> ddown 1i "height"
.PE
```

draws
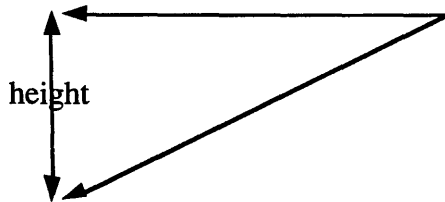
The notation `<->` indicates a two-headed arrow; use `->` for a head on the end and `<-` for one on the start. Lines and arrows are really the same thing; in fact, `arrow` is a synonym for `"line ->"`.

If you don't put any distance after `up`, `down`, and so on, `pic` uses the standard distance:

```
line up right
line down
line down left
line up
```

draws a parallelogram:
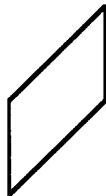
If a set of commands is enclosed in braces {...}, the current position and direction of motion when the group is finished will be exactly where they were when entered. Nothing else is restored.

◆ **Note** There is also a more general way to group objects, using brackets (see "Using Blocks" later in this chapter). ◆

Although objects are normally connected left to right, this can be changed. If you specify a direction as a separate object, subsequent objects will be joined in that direction. Thus,

```
down; ellipse; arrow; circle
```
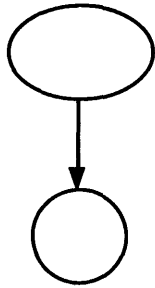
produces

and

```
left; box; arrow; ellipse; arrow; circle
```
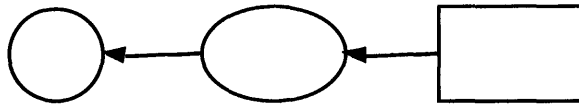
produces

A line may actually be a path; that is, it may consist of segments connected in a direction like this:
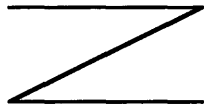
This line was produced by

```
line right 1i then down .5i left 1i\
then right 1i
```

The elements of a path, whether for line or spline, are specified as a series of points, either in absolute terms or by up, down, and so forth. If necessary to disambiguate, the word then can be used to separate components, as in

```
line right then up then left then up
```

This produces

and is not the same as

```
line right up left up
```

which produces

Using blocks

Any sequence of pic statements may be enclosed in brackets [...] to form a block, which is then treated as a single object and manipulated like an ordinary box. For example, the code

```
box "1"
[ box "2"; arrow "3" above; box "4" ] \
with .n at last box.s - (0,0.1)
"thing" at last [].s
```

produces the following picture:
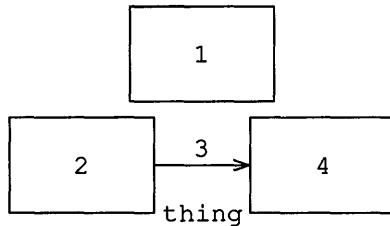
Notice that `last`-type constructs treat blocks as a unit and don't look inside for objects: `last box.s` refers to box 1, not box 2 or 4. You can use `last [ ]`, and so on, just like `last box`.

Blocks have the same compass corners as boxes (determined by the bounding box). You can position a block by placing either an absolute coordinate (like 0,0) or an internal label (like A) at some external point, as in

`[. . .; A: . . .; . . . ] with .A at . . .`

Blocks join with other objects at the center of the appropriate side.

Names of variables and places within a block are local to that block, and thus do not affect variables and places of the same name outside. You can get at the internal place names with constructs like this:

`last [].A`

   or

`B.A`

where `B` is a name attached to a block like so:

`B : [ . . . ; A: . . .; ]`

When combined with `define` statements, blocks provide a reasonable simulation of a procedure mechanism. See "Creating Macros" later in this chapter.

Even though blocks may occur inside of other blocks, you can look only one level deep with qualifiers such as `B.A`. The block `A` may be further qualified so that specifications such as `B.A.sw` refer to the southwest corner of the block named `A`, which is inside block `B`.

For example, the object
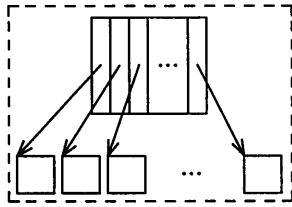
is produced with these specifications:

```
lh = .5i
dh = .02i
dw = .1i
[
Ptr: [
boxht = h; boxwid = dw
A: box
B: box
C: box
box wid 2*boxwid "..."
D: box


Block: [
boxht = 2*dw; boxwid = 2*dw
movewid = 2*dh
A: box; move
B: box; move
C: box; move
box invis "..." wid 2*boxwid; move
D: box
with .t at Ptr.s - (0,h/2)
arrow from Ptr.A to Block.A.nw
arrow from Ptr.B to Block.B.nw
arrow from Ptr.C to Block.C.nw
arrow from Ptr.D to Block.D.nw
]
box dashed ht last [].ht+dw wid last\
[].wid+dw at last []
```

Using the `chop` facility

Sometimes it is desirable to have a line intersect a circle at a point that is not one of the eight compass points `pic` knows about. In such cases, the proper visual effect can be obtained by using the attribute `chop` to chop off part of the line:

```
circle "a"
circle "b" at 1st circle - (0.75i, 1i)
circle "c" at 1st circle + (0.75i, -1i)
line from 1st circle to 2nd circle chop
line from 1st circle to 3rd circle chop
```

This produces
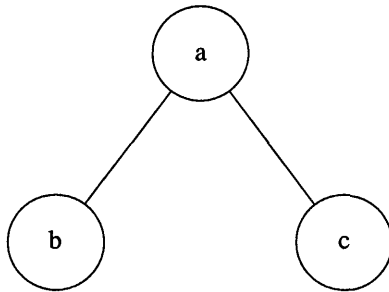
By default, the line is chopped by `circlerad` at each end. This can be changed with the command

```
line ... chop r
```

which chops both ends by *r*, and this specification

```
line ... chop r1 chop r2
```

chops the beginning by *r1* and the end by *r2*.

Another form of positioning refers to a point as a fraction of the way between two other points:

fraction `of the way between` *position1* `and` *position2*

fraction is any expression, and *position1* and *position2* are any positions. You can abbreviate this

fraction < *position1, position2* >

For example,

```
box
arrow right from 1/3 of the way\
between last box.ne and last box.se
arrow right from 2/3 <last box.ne,\
last box.se>
```

produces
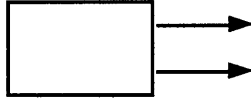
The distance given by *fraction* can be greater than 1 or less than 0.

Creating macros

`pic` provides a rudimentary macro facility, the simple form of which is identical to that in `eqn`:

`define` *name* X *replacement-text* X

This defines *name* to be the *replacement-text*; X is any character that does not appear in the replacement. Any subsequent occurrence of *name* will be replaced by *replacement-text*. Macros with arguments are also available. The replacement text of a macro definition may contain occurrences of $1 through $9; these will be replaced by the corresponding actual arguments when the macro is invoked. The invocation for a macro with arguments is

`name(`*arg1,* *arg2,* `...)`

Nonexistent arguments are replaced by null strings.

As an example, one might define a `square`:

`define square X box ht $1 wid $1 $2 X`

Then

`square(1i, "one" "inch")`

calls for a one-inch square with the obvious label, and

```
square(0.5i)
```

calls for a square with no label:
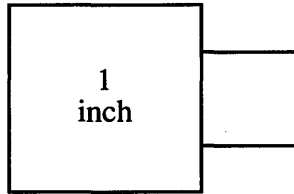
Coordinates like x,y may be enclosed in parentheses, as in (x,y), so they can be included in a macro argument.

Understanding mathematical functions

`pic` provides a number of built-in arithmetic, trigonometric, and random number functions. These are listed in Table 9-3.

Table 9-3 Mathematical functions

Function	Description
`atan2(`e_1`,`e_2`)`	Arctangent of e_1/e_2
`cos(`e`)`	Cosine of e
`int(`e`)`	Integral part of e
`log(`e`)`	Natural logarithm of expression e
`max(`e_1`,`e_2`)`	Maximum of e_1 and e_2
`min(`e_1`,`e_2`)`	Minimum of e_1 and e_2
`rand(`e`)`	Random number from 1 to e
`sin(`e`)`	Sine of e
`sqrt(`e`)`	Square root of e

The arguments to the trigonometric functions (sin, cos, atan2) are assumed to be in radians. All other dimensions are assumed to be in inches. Examples using these functions can be found in "Examples of pic Specifications" later in this chapter.

Understanding loops and conditional statements

Newer versions of pic provide two very useful features: for loops and conditionals with if. An example of the for loop is as follows:

```
.PS
for len=0 to 2 by 0.1 do
    X
                line right len ; line up len
                move left len ; move down len
                move down 0.1
    X
.PE
```

This will produce
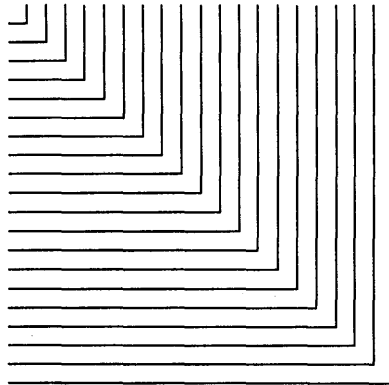
The character x can be replaced by any other unique character; it serves merely to delimit the statements that `pic` will loop through. Also, the increment specifier `by 0.1` may be omitted; if so, the increment specifier defaults to 1.

You may execute `pic` commands conditionally by using the `if` construction. The following example draws 15 boxes at random locations; in addition, all boxes whose length exceeds the height are dashed, while the rest are dotted:
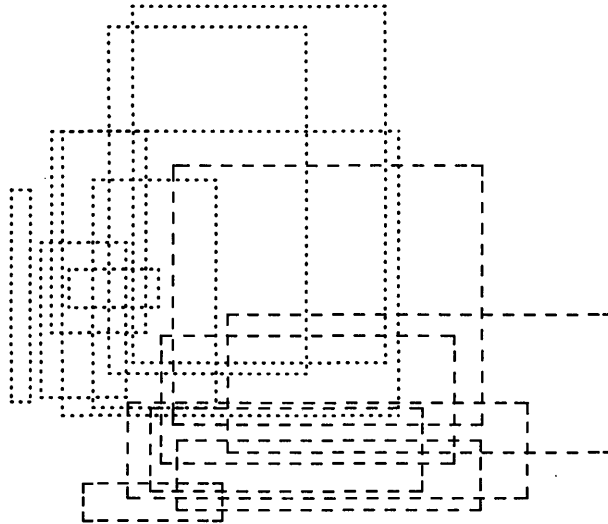
This was specified as

```
.PS
for num = 1 to 15 do
     W
     x = rand(50) /25
     y = rand(50) /25
     if ( x > y ) then
         Y
           box dashed wid x ht y at x,y
         Y else Z
           box dotted wid x ht y at x,y
         Z
W
.PE
```

Understanding expressions

Expressions in `pic` are evaluated in floating point. All numbers representing dimensions are taken to be in inches.

expression:

```
e + e
e - e
e * e
e / e
e % e (modulus)
- e
( e )
variable
number
```

place `.x`
place `.y`
place `.ht`
place `.wid`

Examples of `pic` specifications

Figures 9-2 through 9-9 contain examples of complicated `pic` specifications.
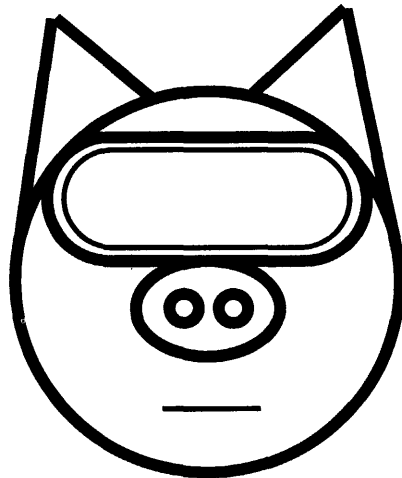
Figure 9-2 Space pig

```
.PS
.ps 100
A: circle radius 1 at 0,0
B: ellipse wid (0.75) height (0.50)\
    with .n at (0,0.1)
C: circle radius (.075)\
    with .e at B.c - (0.05,0)
D: circle radius (.075)\
    with .w at B.c + (0.05,0)
line from (-.97,0.25) to (-.75,1.4)
line from (0.97,0.25) to (0.75,1.4)
line from (-.75,1.4) to (-.25,0.97)
line from (0.75,1.4) to (0.25,0.97)
define goggles\
    @ [ up arc cw rad $1; line right $2;\
    arc cw rad $1; \
    arc cw rad $1; \
    line left $2 ; \
    arc cw rad $1 ] @
.ps 80
E: goggles(0.33, 0.93) with .s at B.n
.ps 40
F: goggles(0.26, 0.90) with .c at E.c
.ps 40
move to (-0.25,-0.675)
line right 0.5
.ps 10
.PE
```

Figure 9-3 Source code for "space pig"
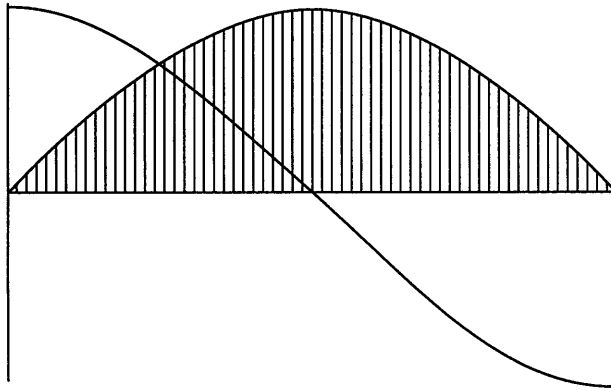
Figure 9-4 Sine and cosine curves

```
.PS
.ps -2
.
pi = atan2(0,-1)
for i = 0 to pi by 0.01 do
X
            "." at i, sin(i)
            "." at i, cos(i)
X
line from (0,-1) to (0,1)
line from (0,0) to (pi,0)
.
for i = 0 to pi by 0.05 do
Y
            line from (i,0) to (i,sin(i)) - (0,.03)
Y
.ps +2
.PE
```

Figure 9-5 Source code for "sine and cosine curves"
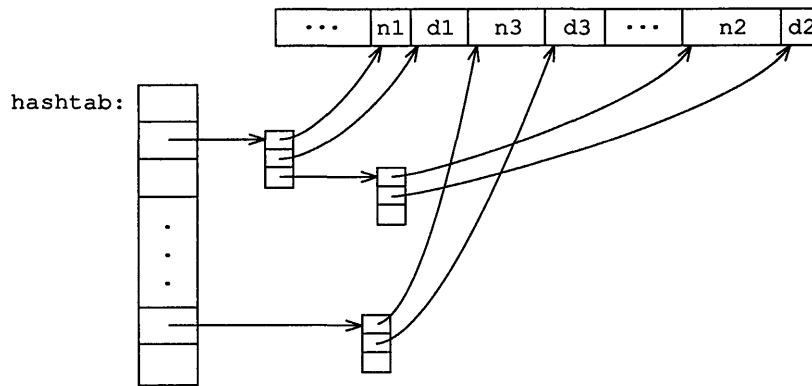
Figure 9-6 File-system diagram

```
.PS
boxht = .2i; boxwid = .3i
down; box; box; box; box ht 3*boxht "." "." "."
L: box; box; box invis wid 2*boxwid "hashtab:"\
            with .e at 1st box .w
right
Start: box wid .5i\
            with .sw at 1st box.ne + (.4i,.2i) "..."
N1: box wid .2i "n1"; D1: box wid .3i "d1"
N3: box wid .4i "n3"; D3: box wid .3i "d3"
box wid .4i "..."
N2: box wid .5i "n2"; D2: box wid .2i "d2"
```

(continued)➡

Figure 9-7 Source code for "file-system diagram"

Figure 9-7 (*continued*)

```
arrow right from 2nd box
ndblock
spline -> right .2i from 3rd last box\
          then to N1.sw + (0.05i,0)
spline -> right .3i from 2nd last box\
          then to D1.sw + (0.05i,0)
arrow right from last box
ndblock
spline -> right .2i\
          from 3rd last box\
          to N2.sw-(0.05i,.2i)\
          to N2.sw+(0.05i,0)
spline -> right .3i from 2nd last box\
          to D2.sw-(0.05i,.2i)
          to D2.sw+(0.05i,0)
arrow right 2*linewid from L
ndblock
spline -> right .2i from 3rd last box\
          to N3.sw + (0.05i,0)
spline -> right .3i from 2nd last box\
          to D3.sw + (0.05i,0)
.PE
```
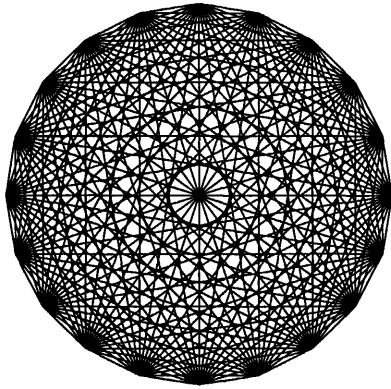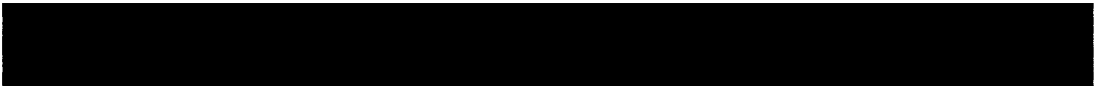
Figure 9-8 Geometric shape

```
.PS
pi = 3.14159; n = 20; r = 1
s = 2*pi/n
for i = 1 to n-1 do
  X
    for j = 1+1 to n do
      Y
              line from r*cos(s*i), r*sin(s*i)\
                to r*cos(s*j), r*sin(s*j)
      Y
  X
.PE
```

Figure 9-9 Source code for "geometric shape"

10 `grap` Graphs
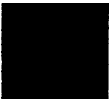
This chapter is a guide to `grap`, a graph-drawing program that allows you to create charts and graphs in your `troff` documents. `grap` operates as a `pic` preprocessor, which means that it reads the description of the graph you specify and produces it as a `pic` drawing.

What is `grap`?

`grap` is a language for describing graphs and charts that are included in documents produced with `troff`. Figures 10-1 and 10-2 are simple examples of the kind of output that you are able to produce using `grap`.

Text
processing
programs

```
sed    
vi     
grap   
pic    
tbl    
egn    
negn   
troff  
nroff  
psdit  
lp     
```

```
I              I              I
0            50000         100000
```

Program size (bytes)
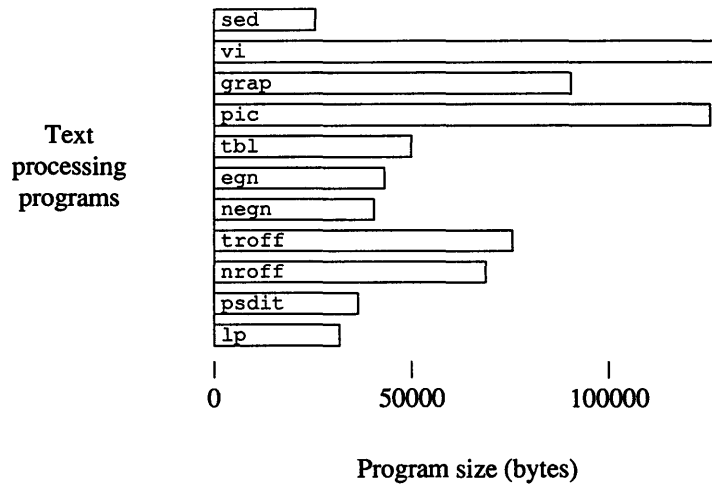
Figure 10-1 A simple graph

Figure 10-1 is a typical bar chart, depicting the relative sizes of some of the A/UX text-processing tools. Figure 10-2, in contrast, is quite a different kind of graph; it gives us a graph of the sine curve over one cycle.
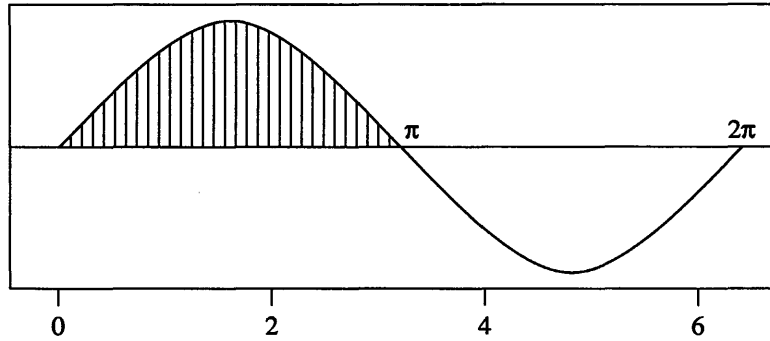
Figure 10-2 A more complicated graph

grap operates as a pic preprocessor, in the same way that pic operates as a troff preprocessor. Graphs are marked in the text by enclosing their descriptions between .G1 and .G2 pairs. The grap preprocessor translates these descriptions into the language understood by pic, which must then be called to translate the grap output into pure troff commands.

This chapter is designed to acquaint you with grap. The grap keywords and commands are introduced largely through examples. A complete reference list of grap syntax is given in the last section of this chapter, "Summary of grap Syntax."

Using grap

grap is usually run with the command line

grap *file* | pic | troff -mm

If equations and tables are also present, you should run grap and pic before eqn and tbl:

grap *file* | pic | tbl | eqn | troff -mm

There are two command-line arguments understood by grap:

-l	Do not include the file containing macro definitions, /usr/lib/dwb/grap.defines. By default, this file is included whenever grap is called.
-T*type*	Set the output device to *type*. Currently supported devices are aps for the Autologic APS-5, and di10 for the Imagen Imprint 10. The default device is aps. In general, however, this argument can be omitted with no ill effects.

Defining the graph format

A graph specification begins with a graph start command (.G1) and concludes with a graph end command (.G2). The .G1 and .G2 commands are used by troff as command delimiters. The general format of grap input is

.G1
chart-specifications
.G2

Individual commands must be separated by newlines or semicolons; a long element may be continued by ending the line with a backslash (\). Comments are introduced by a # and terminated by a newline.

In addition to grap commands, the chart specification can also include troff and pic commands. troff dot commands may be included if they begin a new line; such commands are most useful for changing point sizes in order to get thicker or thinner lines. Included pic commands must be preceded by the keyword pic; this instructs grap to ignore the rest of the line, passing it on to pic.

Specifying charts: Default actions

The following table lists real and projected UNIX operating system-based hardware shipments for the years 1984 to 1990; as the table heading indicates, amounts are in billions of U.S. dollars, and units shipped are in thousands.

	The UNIX market	
Year	Revenues (billions)	Units shipped (thousands)
1984	5.3	127.1
1985	6.5	161.3
1986	7.9	205.0
1987	9.5	265.0
1988	11.3	340.0
1989	13.9	414.0
1990	16.8	485.0

The same data can be entered as a list of numbers using the simplest `grap` specifications. For instance, the following input

```
.G1
1984        5.3        127.1
1985        6.5        161.3
1986        7.9        205.0
1987        9.5        265.0
1988       11.3        340.0
1989       13.9        414.0
1990       16.8        485.0
.G2
```

produces the graph in Figure 10-3.

This chart illustrates many of `grap`'s default actions. First of all, unless instructed otherwise, `grap` will plot the data in a frame that is three inches wide and two inches tall. Also, `grap` automatically supplies ticks indicating the ranges of the data points, drawing them along the left and bottom sides. The ticks are arranged to leave a margin of 7 percent on all sides of the graph. The default plotting tool is the bullet. Finally, `grap` interprets the data in both the second and third columns as belonging to the data in the first column, and (unless told differently) interprets them in the same scale. So `grap` has plotted the yearly system revenues in the same coordinate system as it used to plot the number of units shipped.

Obviously, this chart could stand some improvement. One major failing is the lack of text labeling the various axes and the data points.
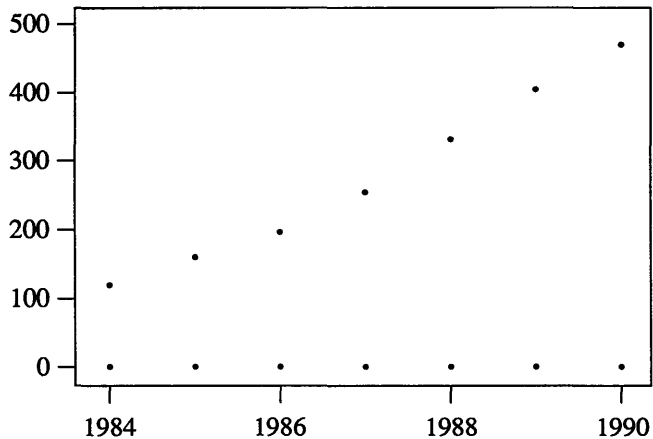
Figure 10-3 The default graph

Also, the bullets look rather lonely plotting the data points. It would be nice to do better, and `grap` provides numerous facilities to override and supplement its default actions. The chart in Figure 10-4 represents the original information more effectively.
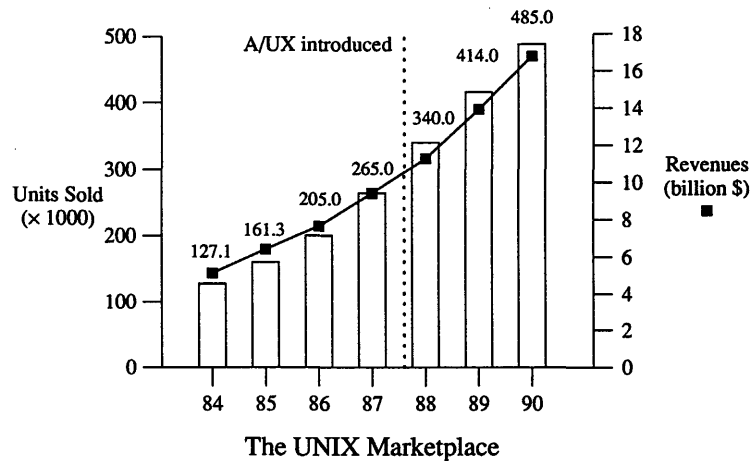
Figure 10-4 A better graph

The following sections provide the information necessary to turn `grap`'s default chart into this more elaborate chart.

Adjusting the frame

Every graph is surrounded by a frame (which may be invisible); this determines the size of the graph. You can adjust the size of the frame with the `grap` command frame. For instance, the command

```
frame ht 3 wid 4
```

will set the height to three inches and the width to four inches. Because `grap` ultimately translates its input into `pic` commands, the largest graph is the largest possible `pic` drawing.

By default, the frame is drawn solid; this can be changed by adding an attribute specifier to the `frame` command. For the moment, disregard the second column of data. So you might have

```
.G1
frame dashed ht 2.5 wid 3.5
1984          127.1
1985          161.3
1986          205.0
1987          265.0
1988          340.0
1989          414.0
1990          485.0
.G2
```

This code produces the graph shown in Figure 10-5.

In addition to `dashed`, other available drawing attributes are `dotted`, `invis`, and `solid`.

You may also specify that only parts of the frame be drawn with a specific attribute. For example, the following is very common:

```
frame ht 3 wid 4 top invis right invis
```

This will draw only the bottom and left sides.
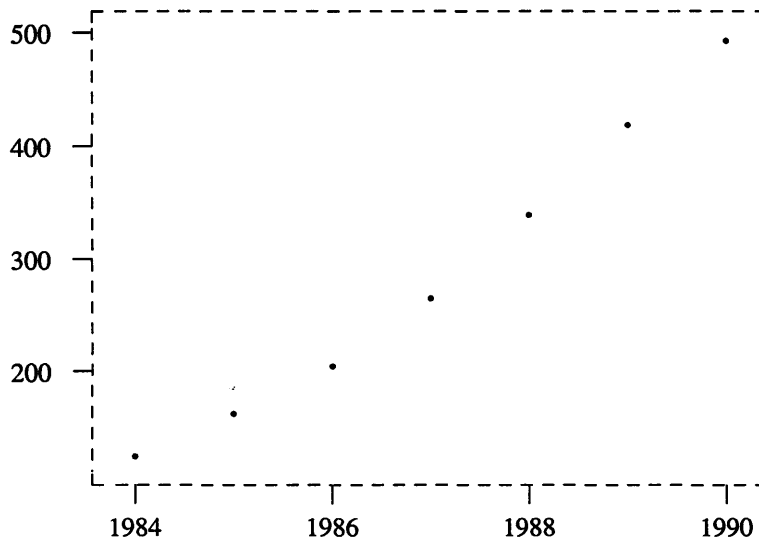
Figure 10-5 A dotted frame

Adding text to a chart

`grap` contains several ways to put text of various sorts into a chart. You have already seen that `grap` automatically supplies ticks on the bottom and left sides indicating the ranges of the data points. More generally, text items can be placed in a chart with the `plot` command. For example, the command

```
plot "A/UX introduced" rjust at 1987.5,300
```

will print the indicated text at the indicated point, right justified. The default action is to center the text item at the specified point. Other positional modifiers are `ljust`, `above`, and `below`. Strings in `grap` are enclosed within double quotation marks, as illustrated. Also, the word `plot` is optional.

Labels can be added to any of the four sides of a chart using the `label` command, for example,

```
label bottom "The 49ers' Season"
```

Multiple text strings are centered one above the others, as with `pic`. If the default placement of the labels is not acceptable, the labels may be shifted in any direction by adding a position modifier:

```
label bottom "The 49ers' Season" down .1
```

This will print the specified text, centered along the bottom of the chart, bumped down one-tenth of an inch. Instead of `down`, the text can also be shifted `up`, `left`, or `right`.

Text items can contain `troff` commands for size and font changes, local motions, and so on, but you should make sure that these are balanced so that the entering state is restored before exiting from the string. So, for example, you might have the following input:

```
plot "\s12The \fB49ers'\fP Season\s0"
```

Adding grid lines to a chart

It is sometimes useful to add grid lines to a chart—to indicate that a certain level has been achieved, to signal important events, or perhaps just to make the chart easier to read. Grid lines are specified with the command `grid`. For example,

```
.G1
frame ht 2.5 wid 3.5 top solid left solid
grid bottom dotted at 1987.5
plot "A/UX introduced" rjust at 1987.5,300
1984        127.1
1985        161.3
1986        205.0
1987        265.0
1988        340.0
1989        414.0
1990        485.0
.G2
```

will produce the graph in Figure 10-6.
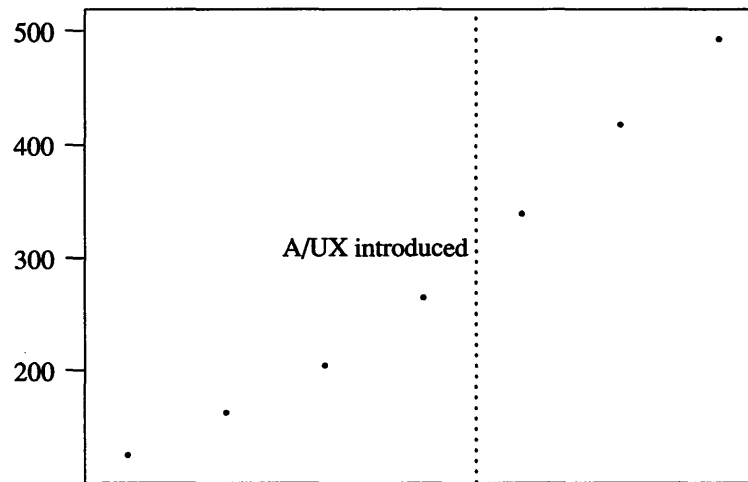
Figure 10-6 Adding grid lines

Using the shell

There are three important ways in which `grap` can interact directly with the A/UX system. It can take input from files located in an A/UX file system, send output to the standard error file, and run arbitrary A/UX commands by passing instructions to the shell.

Instead of presenting your data to `grap` by including it in the chart specification, you can tell `grap` to get some data from a file. This is done with the `copy` command. For example, if the data is stored in a file named `unix.data`, you could simply write the following command:

```
.G1
copy "unix.data"
.G2
```

The result of this graph specification is to produce the default chart given in Figure 10-3. Notice that you had to enclose the name of the file in double quotation marks.

`grap` is also able to send information to the operating system. One way to do this is by using the `print` command. The `print` command sends its argument, either the value of an expression or a string, to the standard error output file. Usually this is the user's terminal screen. For instance, the command sequence

```
.G1
x = 5
print x*7
.G2
```

will result in the value 35 being written on the user's screen. The `print` command is most useful for debugging purposes.

By far the most powerful form of interaction between `grap` and the A/UX system is the `sh` command. The `sh` command passes its arguments (presumably commands) to the A/UX shell; these commands are executed, and control is then passed back to `grap`.

A typical use of the `sh` command is to produce the data that will subsequently be plotted by `grap` using the `copy` command, for example,

```
.G1
sh @ awk -f /tmp/awkscript chap.1 > out @
copy "out"
.G2
```

In this example, `grap` will run the `awk` program using the specified script and redirect the output into a file; this file, `out`, is then copied in and `grap` continues processing the data it has just created. Presumably, the `awk` script generates columns of numbers that `grap` can understand. Note also that there is no reason that this `grap` input could not occur in the file `chap.1` itself.

Creating macros

`grap` provides a rudimentary macro facility, the simple form of which is identical to that in `pic`:

`define` *name* x *replacement-text* x

This defines *name* to be the *replacement-text;* x may be any character that does not appear in the replacement. Any subsequent occurrence of *name* will be replaced by *replacement-text*.

Macros with arguments are also available. The replacement text of a macro definition may contain occurrences of the indicators $1 through $9; these will be replaced by the

corresponding actual arguments when the macro is invoked. The invocation for a macro with arguments is

name(*arg1*, *arg2*, ...)

Nonexistent arguments are replaced by null strings.

Using the `copy thru` construction

`grap` contains a `copy thru` construction, identical to the one in `pic`, that allows the graph data to be interpreted according to the instructions defined earlier in a macro. A typical use of `copy thru` is

```
.G1
define cprint @ circle rad $1 at $2,$3 @
copy "term.data" thru cprint
.G2
```

This will cause `grap` to open the file `term.data` in the current directory and plot a circle of radius determined by the first field at a location determined by the second and third fields.

The data provided to `copy thru` does not need to be taken from a file, nor does the macro need to be predefined. See the entry for `copy` in the "Summary of `grap` Syntax" later in this chapter for a complete list of the possible forms that a `copy thru` construction can take.

Using loops and conditionals

Like `pic`, the `grap` program provides looping and conditional constructions. Looping through a sequence of statements can be achieved with the `for` command. The general form of a `grap` loop is

```
for var = start to end [by step] do
     @ cmds @
```

If the optional *step* specification is omitted, the loop proceeds in increments of 1; also, the character '@' may be replaced by any other character that does not occur in the series of commands *cmds*. In fact, the following form will also work, where the character '@' has been replaced by matching braces:

```
for var = start to end [by step] do

    { cmds }
```

For instance, the curve corresponding to the equation

$$y = x^2$$

can be obtained very easily using the following `grap` instructions:

```
.G1
frame ht 3 wid 3
draw solid
for i = -2 to 2 by 0.1 do
    {
            next at i, i*i
    }
.G2
```

The resulting graph looks like the one in Figure 10-7.

The general form of the `grap` conditional statement is

```
if cond then @ cmds1 @ [else @ cmds2 @]
```

If the condition *cond* is true, then the sequence of commands *cmds1* is executed. If the optional `else` clause is present and if the condition evaluates false, then the sequence of commands *cmds2* is executed; otherwise it is ignored.
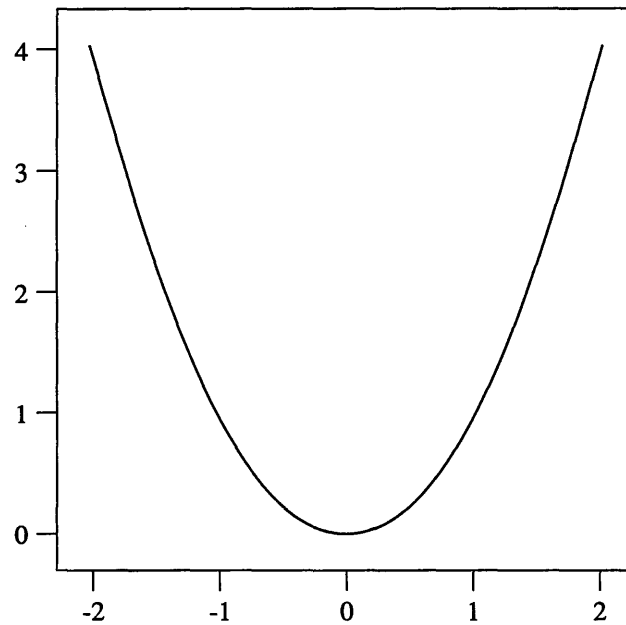
Figure 10-7 Plotting a simple curve

You can add a simple if statement to the previous example to shade in the positive side of the curve:

```
.G1
frame ht 3 wid 3
draw solid
for i = -2 to 2 by 0.1 do
    {
            next at i,  i*i
            if (i >= 0) then
                @ line from 0,i*i to i,i*i @
    }
.G2
```

The resulting graph looks like the one in Figure 10-8.
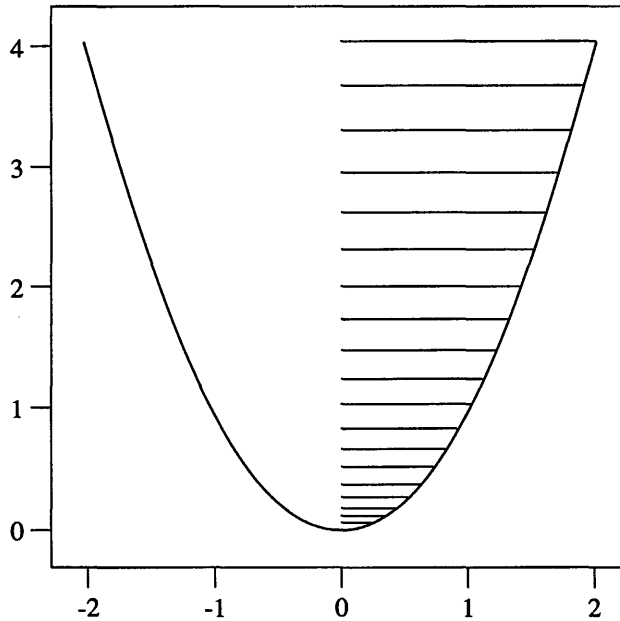
Figure 10-8 Shading part of a curve

Plotting curves

You saw in "Using Loops and Conditionals" earlier in this chapter that the grap language can be used to plot curves from equations as well as from discrete data points. In general, you can use this method to graph any function $y = f(x)$, where $f(x)$ can be expressed using the operators and functions built into grap. The built-in operators are

+	Addition
–	Subtraction
*	Multiplication
/	Division
=	Equality

The built-in functions are

`atan2` (*expr₁,expr₂*)	Arctangent of $expr_1/expr_2$
`cos` (*expr*)	Cosine of expression *expr*
`exp` (*expr*)	Ten to the power *expr*
`int` (*expr*)	Integral part of expression *expr*
`log` (*expr*)	Logarithm base 10 of expression *expr*
`max` (*expr₁,expr₂*)	Maximum of $expr_1$ and $expr_2$
`min` (*expr₁,expr₂*)	Minimum of $expr_1$ and $expr_2$
`rand` (*expr*)	A random number between 1 and *expr*
`sin` (*expr*)	Sine of expression *expr*
`sqrt` (*expr₁*)	Square root of expression *expr*

Consider, for example, the built-in logarithm function `log`. This provides only the base-10 logarithm, but you can define the natural (base-*e*) logarithm if you recall the following simple fact:

$$\log_e (x) = \log_e (10) \times \log_{10} (x)$$

Because $\log_e (10)$ is an easily determinable constant, you can reconstruct the following `grap` macro:

```
define ln @ 2.30258 * log($1) @
```

Furthermore, the function $y = e^x$ is the inverse function of $y = \ln(x)$, so you can graph it by reflecting the graph of the natural logarithm across the diagonal line $y = x$. So you have

```
.G1
define ln @ 2.30258 * log($1) @
frame ht 4 wid 4
draw Nat solid
draw Ten dotted
draw Exp solid

for i = 0.5 to 5 by 0.1 do
  {
              next Nat at i, ln(i)
              next Ten at i, log(i)
              next Exp at ln(i), i
  }
```

```
line dashed from 0,0 to 4,4
"$y~=~e sup x $" at 0.0, 2.00
"$y~=~ln (x) $" at 2.0, 1.0
"$y~=~log (x) $" at 3.0, 0.65
"$y~=~x $" at 4.5, 4.5
.G2
```

This yields the graph shown in Figure 10-9.
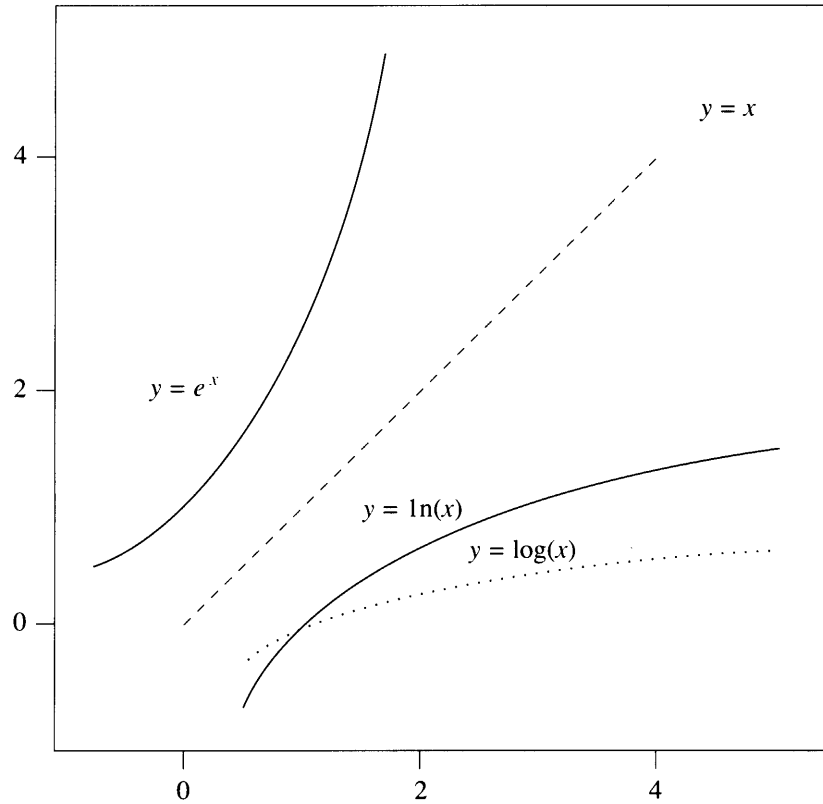
Figure 10-9 Logarithmic and exponential functions

Using polar coordinates

Some curves are more easily described using polar coordinate equations than using Cartesian rectangular coordinates. For example, the polar equation of a circle with its center located at the origin is simply $r = a$, for some constant a, whereas the rectangular equation is the somewhat more complicated $x^2 + y^2 = a^2$. Even though `grap` does not contain primitives for handling polar equations, it is relatively straightforward to graph some equations expressed in polar form, $r = f(\theta)$.

To see this, consider the following simple relationship between the sides of a right triangle:
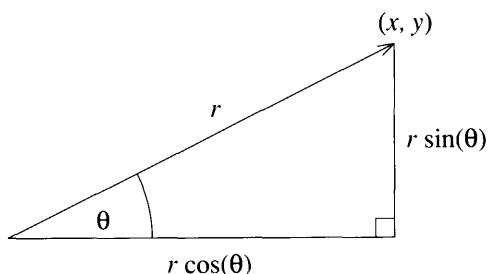
You notice the following two facts:

$x = r\cos(\theta)$

$y = r\cos(\theta)$

You can therefore graph the curve $r = f(\theta)$ by plotting the sets of point x,y that satisfy the equations

$x = f(\theta) \times \cos(\theta)$

$y = f(\theta) \times \cos(\theta)$

For example, suppose that you want to graph the Spiral of Archimedes, $r = \theta$ The following `grap` input will do nicely:

```
.G1
frame ht 3.5 wid 2
label bot "Spiral of Archimedes"  "$r~=~theta$"
pi = 3.14159

for i = 0 to 3*pi/2 by 0.1 do
  {
    next at i*cos(i),  i*sin(i)
  }
.G2
```

This yields the graph in Figure 10-10.

Spiral of Archimedes
$r = \theta$
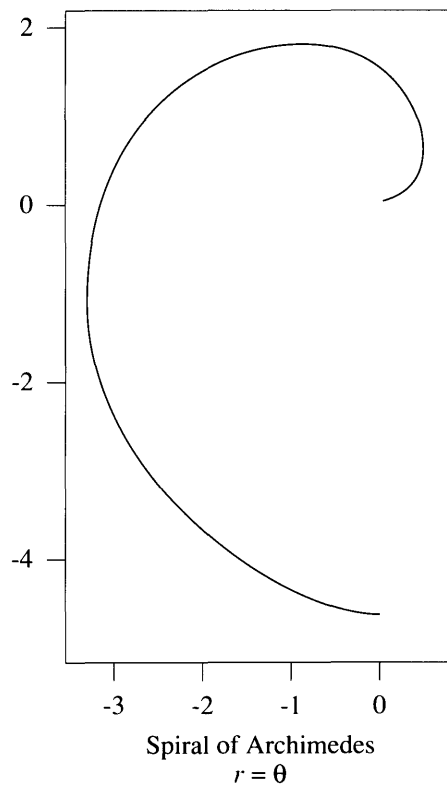
Figure 10-10 Plotting a polar equation

Similarly, you can give the following code to generate the graph of the cardioid:

```
.G1
frame invis ht 3.0 wid 3.5
grid bottom dotted at 0
grid left dotted at 0
ticks off
label bottom "Cardioid" "$r~=~1~-~ cos ( theta ) $"
"X" at 1,0.25
"Y" at 0.25,1.5
pi = 3.14159
for i = 0 to 2*pi by 0.07 do
    {
      next at (1-cos(i))*cos(i),  (1-cos(i))*sin(i)
    }
.G2
```

This code yields the graph in Figure 10-11.
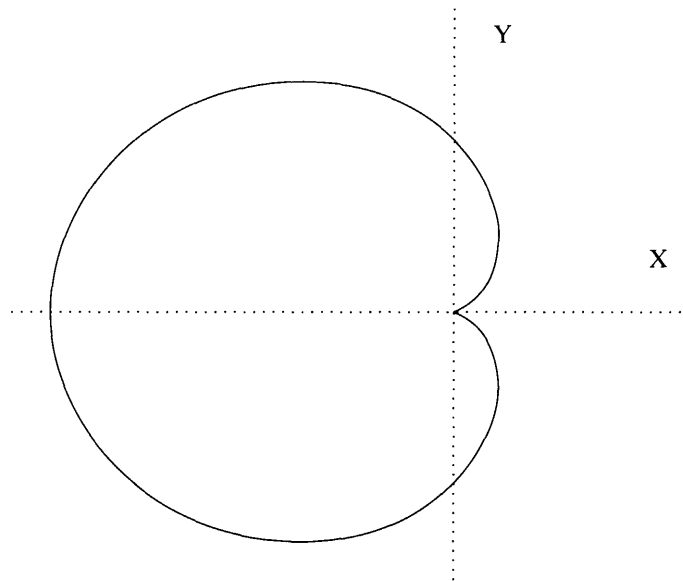
Cardioid
$$r = 1 - \cos(\theta)$$

Figure 10-11 A second polar equation

As a final example of the power of this method, consider how easy it is to graph a circle using polar coordinates. You noted that the polar equation of a circle centered at the origin is just $r = a$. The necessary transformations into x,y pairs are therefore

$$x = a \times \cos(\theta)$$
$$y = a \times \sin(\theta)$$

So, the following code produces the circle of radius 1 shown in Figure 10-12.

```
.G1
frame invis ht 3 wid 3
pi = 3.14159
for i = 0 to 2*pi by 0.05 do
   {
      next at cos(i), sin(i)
   }
next at 1,0 # close up graph
.G2
```

The interested reader should attempt to recreate this graph without using polar coordinates. (No fair using the `pic` built-in `circle`!)
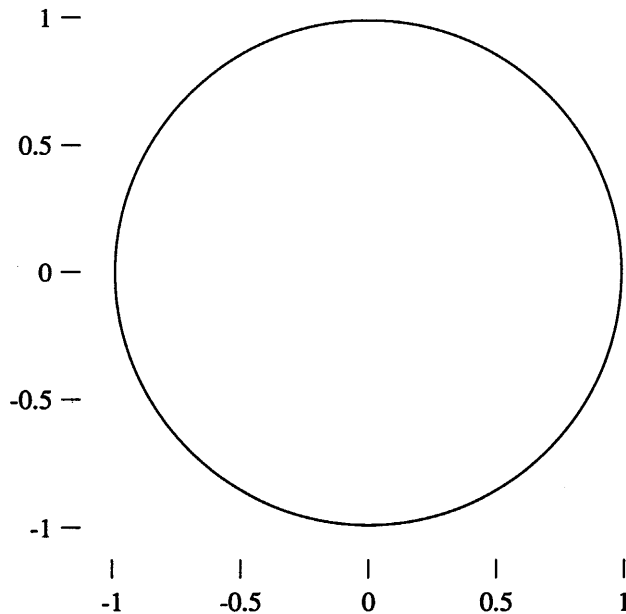
Figure 10-12 A `grap` circle

Using equally scaled axes

You will notice that `grap` automatically calculates the bounds of the curve being graphed and scales the coordinate axes in such a way as to fit the graph into the space available (either the size requested using the `frame` command or the default size). This means that the axes are almost never drawn according to the same scale. For graphs of discrete data this is not generally a problem, but graphs of curves and functions are often misleading unless drawn with axes scaled identically.

There is an easy way to get `grap` to produce equally scaled axes. The `frame` and `coord` statements can be used to specify that the size and coordinate ranges for both axes be identical. For instance, the following `grap` instructions will ensure that the curve looks the way you expect:

```
.G1
frame ht 3 wid 3
coord x 0,10 y 0,10
draw solid
for i from 0.1 to 10 by 0.05 do
   {
     next at i, 1/i
   }
.G2
```

This yields the graph in Figure 10-13.
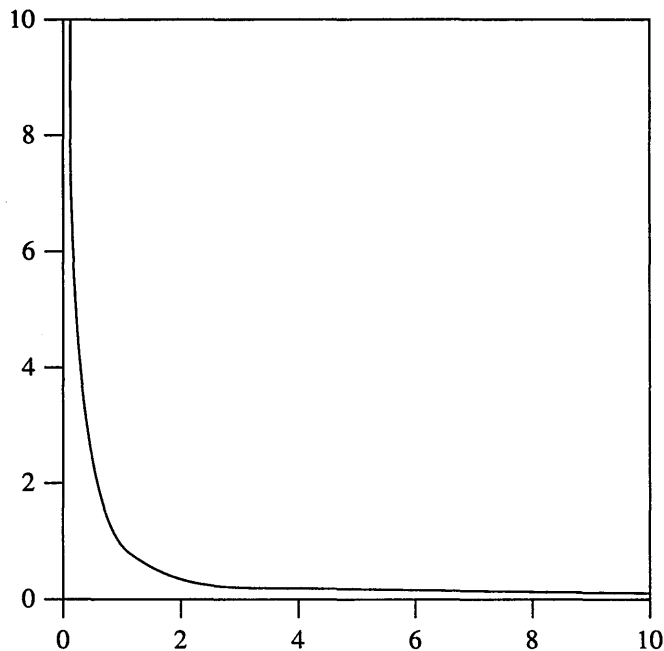
Figure 10-13 Equally scaled axes

You will notice that obtaining equally scaled axes via the `coord` command demands that you have some previous idea of what the bounds of the function are likely to be. If you genuinely have no firm idea what the resulting graph is going to look like, you can still ensure equally scaled axes in the following way: while plotting the set of points x,y over some interval, also plot the set of points y,x *invisibly*. This has the effect of plotting the *inverse* function $y = f^{-1}(x)$, thereby guaranteeing that the largest x value is the same as the largest y value.

To illustrate this second method of producing equal axes, suppose that you want to graph the curve $y = x^3$. You can give the following code, which does not use the `coord` command:

```
.G1
frame ht 3 wid 3
draw Real solid
draw Hack invis
for i from 0 to 3 by 0.1 do
    {
        next Real at i, i*i*i
n       ext Hack at i*i*i, i
    }
.G2
```

This will produce the graph shown in Figure 10-14.
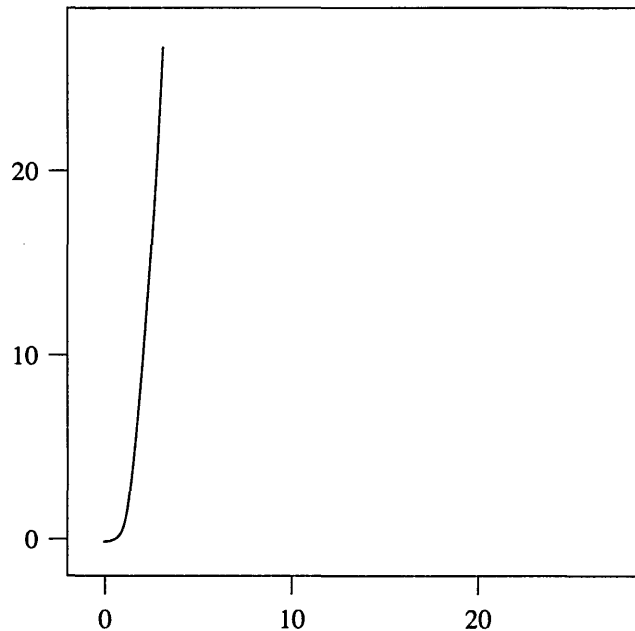
Figure 10-14 Equally scaled axes without `coord`

Plotting curves from data points

Sometimes it is not possible to reduce an equation to the rectangular form $y = f(x)$ or to the polar form $r = f(\theta)$. In such a case, it is still possible to obtain a graph of the function using `grap`. For example, to find a graph of the equation

$$x^8 = (x^2 + y^2)^3.$$

you could write a simple program to generate a set of data points on the curve between 0 and 4, as shown in Figure 10-15.

points

```
#include <stdio.h>
#define STEP 0.01 /* step size between points */
#define MARG 0.01 /* margin of closeness */
#define approx(a,b) ((a>=(1.0-MARG)*b)&&(a<=(1.0+MARG)*b))

main()
{
  float x, y;
  for ( x = 0.0 ; x <= 4.0 ; x += STEP )
      for ( y = 0.0 ; y <= 4.0 ; y += STEP )
          if ( on_curve(x,y) )
              printf("%4.3f %4.3f\n", x, y);
  exit(0);
}


on_curve(fx,fy)
float fx, fy;
{
  if ( approx( fx*fx*fx*fx*fx*fx*fx*fx ,
               ((fx*fx)+(fy*fy))
             *((fx*fx)+(fy*fy))
             *((fx*fx)+(fy*fy)) ) )
      return(1);
  else
      return(0);
}
```

Figure 10-15 Sample C program to generate data

When this program is compiled and run, it will generate a list of data points. If the output of the command is redirected into the file `curve.data`, the following `grap` commands will give you the graph you want:

```
.G1
frame ht 3 wid 3
coord x 0,8 y -4,4
draw Pos solid
draw Neg solid
copy "curve.data" thru @ next Pos at $1,$2
                            next Neg at $1,-$2 @
.G2
```

This yields the graph shown in Figure 10-16.
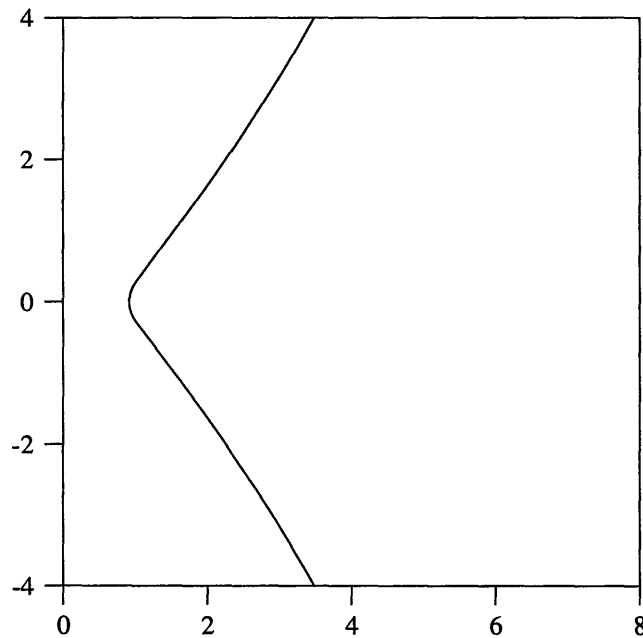
Figure 10-16 Plotting a curve from data points

Summary of `grap` syntax

`grap` is a `pic` preprocessor designed for drawing charts and graphs and including them in documents formatted with `troff`. The general command line is

`grap` *files* `| pic | ... | troff ...`

Graph specifications are included between `.G1` and `.G2` pairs and may include the following commands:

.anything (at beginning of line)
Copy this line untouched. Hence, `troff` commands may be interspersed among `grap` commands.

 # anything
The symbol # is a comment indicator; anything following this symbol on a line will be ignored by `grap`. You can also use the `troff` comment indicator .\" at the beginning of a line to include comments in a graph specification.

`coord` [*dataset*] `x` *min,max* `y` *min,max* [`log x`] [`log y`]
Set the range of the `x` and `y` coordinate axes to run from *min* to *max*. This command overrides the default axis scaling and may result in the loss of data points that do not fit into the specified range. Addition of the optional `log` indicator will result in logarithmic scaling of the specified axis. The default *dataset* is the one currently active.

`copy` *"file"*
Include the file *file* at this point.

`copy` *"file"* `thru` *name* `until` *"str"*
Copy the data from file *file* through macro *name* until the first occurrence of the string *str* is encountered.

`copy thru` *name*
Pass the rest of the input for this graph (that is, until the next `.G2`) through the macro *name*, breaking the line into fields that are passed as arguments to the macro. Fields are delimited by white space, except for white space enclosed by string delimiters, "...". The macro *name* can be replaced by an in-place macro.

`copy thru` *name* `until` *"str"*

As above, except that the copying ends when the first occurrence of the string *str* is found at the beginning of an input line.

`define` *name* `x` *anything* `x`

Define a `grap` macro: replace all subsequent occurrences of *name* by *anything*. If the string *anything* contains any of the sequences $1, $2, ..., $9, they are replaced by the first, second, ..., ninth arguments enclosed in parentheses following *name*. The file `/usr/lib/dwb/grap.defines` contains several macro definitions, and it is included in all files processed by `grap` if it exists (unless the `-l` command line option is specified).

`draw` [*dataset*] *attrib* [*"str"*]

Set the attribute to be used in drawing the graph of data set *dataset* to *attrib*. If the optional string *str* is added, this string will appear at each point plotted.

`for` *var* = *start* `to` *end* [`by` *step*] `do` @ *cmds* @

Run the specified list of commands *cmds* for all the values between *start* and *end*, taken in steps of *step*. If the `by` clause is omitted, steps of 1 are taken. The assignment operator = can be replaced by the keyword `from`.

`frame` [*attrib*] `ht` *h* `wid` *w* [*side attrib*]

Set the frame surrounding the graph to the specified height *h* and width *w*. The default size is 2 inches high and 3 inches wide. You may set the drawing mode *attrib* for the entire frame or for each of the sides (`top`, `bot`, `left`, and `right`) to any one of the attributes `dotted`, `dashed`, `invis`, and `solid`. The default attribute is `solid`.

`graph` *Name pos*

Begin a new frame *Name* for subsequent plotting, placing the frame at the specified *pos*. The position *pos* must be in a form recognizable by `pic`, for instance,

`graph New with .s at Old.n`

The name of the graph *Name* must be capitalized, in accordance with the input syntax for `pic`.

`grid` *side attrib* `at` [*dataset*] *expr*
Draw grid lines perpendicular to the specified side *side* at the value of expression *expr.*
The line is drawn with attribute *attrib,* which is by default `solid`. There may be more
than one expression, and grid lines and labels of incremental steps are available as with
the `ticks` command.

`if` *cond* `then @` *cmds1* `@` [`else @` *cmds2* `@`]
Run the commands *cmds1* if the specified condition *cond* is true. If the condition
evaluates false, and if the optional `else` clause is present, then run commands *cmds2.*

`label` *side* `"`*str*`"` [*"str"*] [*pos expr*]
Use string *str* as a label on the specified side *side.* The default side is the bottom. There
may be any number of strings, which are centered one above the others. In addition, a
label specification may include an optional position *pos* to shift the default position of the
label. The specifier *pos* may be `up`, `down`, `left`, or `right` and must be followed by an
expression indicating the amount of position shift in the specified direction.

`line from` *pt* `to` *pt* [*attrib*]
Draw a line, using the specified attribute *attrib,* from the first point *pt* to the second. The
default attribute is `solid`. Also, the keyword `line` may be replaced by `arrow`.

`next` [*dataset*] `at` *pt* [*attrib*]
Plot the next data point for data set *dataset* at point *pt,* connecting that point with
previous points by a line of attribute *attrib.*

`new` [*dataset*] *attrib* [*"str"*]
Set the attribute to be used in drawing the graph of the data set *dataset* to *attrib,* and
disconnect the subsequent data points from any preceding ones. If the optional string *str*
is added, this string will appear at each point plotted.

number-list

Unless copied through a macro (see "Using the `copy thru` Construction" earlier in this chapter), treat a list of numbers as follows. A single column of numbers (one number per line) is interpreted as a list of ordinates (y values) for the abscissae (x values) 1, 2, 3, ... Multicolumn lists are treated as a single words; a line of the form

x y1 y2 y3 ...

will result in plotting the points $(x,y1)$, $(x,y2)$, $(x,y3)$, and so on.

`pic` *anything*   Pass the remainder of the input line to `pic`, removing any leading white space. The input *anything* cannot contain newlines.

`plot` *"str"*[*loc*] `at` *pt*

Place string *str* at point *pt*. The optional location *loc* can be any one of the modifiers `rjust`, `ljust`, `above`, and `below`. Also, the keyword `plot` may be omitted altogether.

`point` [*dataset*] *expr,expr*

Map the point determined by the values of the two listed expressions to the specified *dataset*. The default data set is the current.

`print` *expr* or `print` *"str"*

Print the value of expression *expr* or the string *str* on the standard error file. This is most useful as a debugging tool.

`sh` @ *anything* @

Pass everything between the enclosing @ characters to the A/UX shell. The character @ can be replaced by any other character. Also, newlines may be included in the string *anything*.

`ticks` *side dir* at [*dataset*] *expr* [*"str"*] [, *expr "str"*]
Draw ticks on the specified side *side* at *expr,* using the optional string *str* as a label. More than one expression and label can be listed, separated from the preceding ones by a comma. Direction *dir* may be either `in` or `out`, indicating the direction the ticks are drawn (the default direction is `out`). The strings specified as labels may contain format specifiers of the form %f*n.m,* which are interpreted as with the C-language function `printf`. See `printf`(3) in the *A/UX Programmer's Reference* for details.

`ticks` *side dir* `from` *m* `to` *n* [`by` *step*] [*"str"*]
Draw ticks on the specified side *side* beginning at value *m* and continuing to value *n* in steps of size *step,* using the optional string *str* as a label. The step size *step* may be preceded by an optional + or - to obtain additive increments or decrements, or by an optional * or / to obtain multiplicative increments or decrements. If the step specifier is omitted, steps of size 1 are used. If no ticks are requested, they will be supplied automatically, although this can be suppressed with the command `ticks off`. A margin of 7 percent is left on each side of a graph; the margin can be adjusted with the command

margin = *expr*

var = *expr*
Set variable *var* to the value of expression *expr.*

11 Related Tools

The tools described in this chapter supplement the text-processing programs described elsewhere in this book. This chapter is intended as a short reference to these additional tools. For complete information on each command, refer to the *A/UX Command Reference*.

What are the other text preprocessors?

Preprocessors operate with text formatters to produce specialized forms of output, such as tables, equations, and line drawings. Preprocessor data is converted to `troff` (or `nroff`) commands, and then this output is passed on to the formatter for further processing.

`tbl`, `eqn`, and `pic` are the most commonly used A/UX preprocessors. The following is a brief description of some lesser-known A/UX tools.

Preparing constant-width text

Text typeset in constant-width (CW) font resembles the output of terminals and line printers. All characters are the same width. CW font is used most often to show examples of computer output.

CW font contains a nonstandard set of characters with character and interword spacing different from that of standard `troff` fonts, such as Times Roman. Documents using the CW font must be preprocessed.

See `cw(1)` in the *A/UX Command Reference* for more information.

Numbering lines

The `nl` program is a line-numbering filter. It reads lines from a named file (or from standard input if no file is named) and reproduces the lines on the standard output. Lines are numbered on the left side of the page.

`nl` processes your text in "logical pages." Line numbering is reset at the start of each logical page. A logical page consists of a header, a body, and a footer section. Different line-numbering options are available independently for each of these sections. For example, you may specify that you do not want header or footer lines numbered. (The default is not to number either header or footer lines.)

To specify the start of each logical page section, use the following default delimiters, which appear at the line preceding the start of the section:

```
\:\:\:        Header
\:\:          Body
\:            Footer
```

Thus, for general purposes \ and : are considered to be the delimiter characters, as they are repeated and joined to form the actual delimiters.

You may specify new delimiter characters by use of the –d flag option. For example, in the command

```
nl -v5 -i5 -d!+ test.file
```

the delimiter characters are changed to !+. The entire command instructs nl to number test.file starting at line number 5 (-v5), with an increment of 5 (-i5). (The default is to begin numbering at line 1 and to use an increment of 1.)

For a complete description of the available options, see nl(1).

Translating characters

The tr program translates characters in a file. It takes two string arguments. Any characters found in the first string are replaced by the equivalent characters in the second string.

For example, suppose you want to convert all uppercase characters in a file to lowercase. You can do this with the command

```
tr "[A-Z]" "[a-z]" < upper.file > lower.file
```

where "[A-Z]" is the first string, "[a-z]" is the second string, upper.file is the original file, and lower.file is the translated file. The double quotation marks and brackets are necessary to distinguish ranges from regular strings. If they are omitted, only the characters A, -, and Z will be translated to lowercase.

You can also use tr to delete character strings. For example, if you want to remove all numeric characters from a file, you may use the command

```
tr -d 0-9 < num.file > unnum.file
```

With the –d option, you specify only one string, and tr deletes members of it wherever they occur. Ranges do not need special treatment with this option.

See tr(1) for more information.

Single-spacing a document

`ssp` removes extra blank lines from a file and causes all output to be single-spaced. You may use it either directly on a file:

```
ssp file > out.file
```

or as a filter following text formatting:

```
troff -mm file | ssp > out.file
```

See `ssp`(1) for more information.

Changing the format of a text file

The `newform` program allows you to change the format of a text file. You may change tab characters to spaces or spaces to tabs. You may define a standard line length, and if your input exceeds that length, you may designate that *n* characters be removed, from either the beginning or the end of each line. If your input lines are shorter than the designated line length, you may choose the number of characters to append or prefix to each line. For example, given `test.file`—a file with lines consisting of leading digits, one or more tabs, and then text—the command

```
newform -s -i -e -a test.file > out.file
```

converts it to a file (`output.file`) with lines beginning with text (`-s`), all tabs expanded to spaces (`-i`), each line padded (or truncated) with spaces to fit 72-column format (`-e`), and the leading digits (which were stripped away with the `-s` option) appended after column 73 (`-a`).

See `newform`(1) for more information.

Printing Greek characters

`greek` is an `nroff` filter that permits you to produce an approximation of the Greek alphabet on output devices not normally able to print nonstandard characters.

The file `/usr/pub/greek` contains the default Greek characters produced by `nroff`. The `greek` filter reinterprets this character set (as well as the default reverse and half-line motions) to permit use on a variety of terminals. The special characters are simulated by overstriking.

Your own terminal type may be specified after the `-T` flag option. Thus the command

```
nroff -mm test.file | greek -Tterm
```

(where *term* specifies the output device) formats `test.file` and filters the output through `greek`. (If no `-T` argument is given, `greek` attempts to use the environment variable `$TERM`.)

greek recognizes only certain terminal types. To view the list of recognized terminal types, see `greek`(1).

Creating underlines for your terminal

The `ul` program translates underscore characters to a sequence that simulates underlining. The actual sequence depends on the options supported by your terminal. Some terminals produce reverse video to indicate underlining; others do actual underlining. If your terminal cannot interpret underscores, `ul` behaves like `cat`(1) and simply displays the file on your screen.

You may specify the terminal type after a `-t`. This is the most reliable way to obtain underlining as such, if your terminal can do it. If no type is specified, `ul` will try to determine it from the environment and may consult `/etc/termcap` to learn how to underline.

Thus the command

```
nroff -mm test.file | ul -t term
```

(where *term* is the terminal type) will format `test.file` and filter the result through `ul` to produce underlining wherever `test.file` had lines preceded by

```
.ul
```

or had `troff` requests for italics.

See `ul`(1) for more information.

Stripping out reverse line feeds

The `col` program allows you to print files that contain reverse line feeds and forward and reverse half-line feeds on output devices that cannot handle reverse movements.

`col` filters out the reverse line feeds generated by the `.rt` (return) `nroff` request, some `eqn` output, `tbl` output, and other multicolumn output. In addition to removing reverse line feeds, the `col` program filters out other nonprinting characters. You can then print your formatted file on simple printing devices.

To run `col` on a multicolumn `nroff` document, use the command

```
nroff -mm test.file | col > output.file
```

See `col`(1) for a complete description of the use of this command.

Using a macro package to typeset viewgraphs and slides

You may use the `mv` macros to prepare typeset-quality viewgraphs and slides. Viewgraphs can be prepared in a variety of dimensions, as well as 35-mm slides and 2-by-2-inch "super slides." A few macros perform most of the formatting tasks needed in making transparencies, and you may use all of the facilities of `nroff`, `troff`, `eqn`, `tbl`, and `cw` for the more difficult tasks. See `mv`(5) for a complete list of the available macros.

To run your text files prepared with `mv`, use the `mvt` command

```
mvt file > out.file
```

Options are provided to call `tbl` and `eqn`, and the proper pipelines and required arguments for `troff` are generated automatically.

See `mmt`(1) and `mvt`(1) in *A/UX Command Reference* and `mv`(5) in *A/UX Programmer's Reference* for more information.

Using special tools for the manual pages

The A/UX manual pages found in the *A/UX Command Reference, A/UX Programmer's Reference,* and *A/UX System Administrator's Reference* contain descriptions of all commands and maintenance procedures contained in the A/UX system. The manual pages are produced according to strict formatting conventions with the `man` macros.

Creating a manual page

The man macros produce standardized manual entries. You use the man macro package to create manual pages in the same way that you use the mm macro package to produce text.

To produce your own manual pages, follow the instructions in man(5).

Reading online manual entries

The man command locates and prints a requested manual entry. The manual page can be viewed on your terminal screen (the default) or can be printed on your printer.

For example, to produce the manual page grep(1) for terminal viewing, enter the following:

```
man 1 grep
```

The 1 is the section number of the manual. It alerts the system to search through section 1. If the section number is not specified, the entire manual set (sections 1 through 8) is searched.

See man(1) for more information.

Creating a permuted index

The permuted index in the *A/UX Command Reference, A/UX System Administrator's Reference,* and *A/UX Programmer's Reference* is produced with the ptx command. The permuted index presents a sorted alphabetic listing of keywords contained in the command descriptions.

It works in three stages:

1. It generates one line for each keyword in an input line and then rotates the keyword to the front of the line.

2. It alphabetically sorts the permuted file.

3. It rotates the sorted lines and places the keyword in the middle of the line.

You can then scan the center column of the permuted index for the keywords.

For flag options and formatting information, see ptx(1) and mptx(5).

Checking your work before you format it

The following tools check your work before you process it. With each command, output can be either written to standard output (the default) or redirected to a file.

Checking your spelling

The `spell` program checks the words in your document against an online dictionary and then reports those words not found. You can instruct `spell` to verify either American or British spelling. You can also stipulate a file (with the `+local.file` flag option) of words not in the online dictionary for `spell` to use as well. In this case, `+local.file` must be sorted, with one word per line.

To run `spell`, enter

```
spell test.file > spell.list
```

This checks the spelling of words in `test.file` and puts dubious ones in `spell.list`.

◆ **Note** Proper names and technical terms appear in the `spell` output (unless you include them in your `+local.file`). Often, you will have to edit these out of your `spell.list` before using it to correct your files. ◆

For complete instructions on using the spell program, see `spell(1)`.

Checking your writing style

You can check certain aspects of your writing style with the `style` program. It gives the use (by percentage) of various grammatical forms, and it reports on readability, sentence length and structure, word length and usage, and types of verbs used.

Although such statistics may seem superficial, they can still be of use. `style` is particularly useful for comparing two documents or seeing if you are overusing a particular grammatical form.

To run `style`, enter the following:

```
style file
```

Checking your document's clarity

The `diction` program finds sentences in a document that are overused or poorly constructed. It compares what is in your document against a database of bad phrases and reports any matches.

To run `diction`, enter the following:

```
diction < file
```

See `diction`(1) for additional information.

Checking your `eqn` commands

`checkeq` looks for missing or unbalanced `eqn` delimiters (usually $$) or `.EQ` and `.EN` pairs. It especially looks for mixtures of these, which would confuse `eqn`; thus the output

```
$$ within .EQ .. .EN , line n
```

indicates that inline delimiters were used within a displayed equation.

To run `checkeq`, enter

```
checkeq file
```

Not all output lines flag errors directly. The diagnostic

```
$$ delims, line n
```

does not report an error. It states that inline delimiters ($$) were turned on at line n. If the delimiters change, this will also be reported. Then, if they are changed to another symbol or if they are left off for a long time, this will be apparent from the output.

◆ **Note** Do not set delimiters to `##` and then use `eqn` within tables. `tbl` uses `#` characters internally and may not be able to function if `eqn` uses them as well. ◆

If you need to use the dollar sign itself, you can use the following `eqn` definition at the top of your file:

```
.EQ
define dol 'roman "$"'
delim $$
.EN
```

Because the single dollar sign appears in the file before the delimiters were turned on, this usage will not cause an error to be reported by `checkeq`. Then, to use this defined term, enter

```
$dol$
```

to produce $. The dollar signs match, and no error is flagged.

◆ **Note** If you use the mm macros, you should use `checkmm` instead of `checkeq`. The `checkmm` program incorporates all the features of `checkeq`. ◆

Checking your mm commands

`checkmm` checks for inconsistent use of the mm macros. It finds unmatched pairs of macros, unmatched size and font changes, and unbalanced `.EQ.`/`.EN` pairs. If you use `checkmm`, you do not have to use `checkeq` as well.

To run `checkmm`, enter the following:

```
checkmm file
```

For more information on options and syntax, see mm(1).

Checking your ms commands

You can check your ms documents for formatting errors with the `checknr` program. `checknr` examines your file and reports any unrecognized macros or unbalanced macro constructions. For example, it will find any `.DS` commands that are not terminated with `.DE`, or it will verify that each `.RS` command has a corresponding `.RE` command.

To run `checknr`, enter the following:

```
checknr file
```

Any discrepancies are written to the standard output. Or, if you prefer, you can direct the output from `checknr` to a file so you can examine it later:

```
checknr file > output-file
```

For more detailed instructions on using this program, refer to `checknr`(1) in *A/UX Command Reference.*

Checking your cw commands

You can use `checkcw` on files to be processed with `cw`. `checkcw` finds unbalanced left and right delimiters, as well as `.CW`/`.CN` pairs.

See `cw`(1) in *A/UX Command Reference* for more information.

Glossary

adjust To add small amounts of space between words in a filled text line so that the line of output text is the desired line length.

argument Used in a command line and placed after the command to specify what the command should act upon.

break Printing of a partially filled output line.

command Sets parameters or calls out special characters.

comment An informative remark embedded in text but not intended for printing. You can include a comment at the end of a line by prefacing it with \.You can include a comment in a file as a line by itself by beginning the line with . \.

control character A period or single quotation mark that calls out a command.

control lines Sometimes called "dot commands," they are interspersed with text lines and set parameters or otherwise control subsequent processing. They begin with a period or an acute accent, followed by a one- or two-character name that specifies a basic request or the substitution of a user-defined macro in place of the control line.

display A block of text that is to be kept on one page. The relevant text is enclosed within the .DS and .DE macros. By default, the text lines are not filled or adjusted, but you can override this by providing an argument to the .DS macro.

diversion A mechanism provided by the troff formatter to store a block of input text for a period of time in order to determine its size and whether it will fit on the current page before actually printing it, for example, footnotes or text between the macros .KS and .KE that is not to be split across a page boundary (as for a figure or table).

em Used to specify a width approximately equal to the size of the letter m in the current font and point size.

en Half of an em.

eqn A mathematical equation-formatting preprocessor for troff that produces typeset-quality mathematical text. eqn converts mathematical input into troff commands, and the resulting output is passed directly to the formatter for further processing. Mathematical expressions are entered by beginning and ending each with the delimiters .EQ and .EN. Inline equations may be included in text if they are enclosed in delimiters, which are defined at the beginning of the text file.

escape character (\), followed by a command name anywhere in a line. The escape character introduces sequences that cause the following character to mean another character or signals the formatter to treat the sequence as a command and not text. It should not be confused with the control character Esc of the same name.

field A string of characters separated from other strings by blanks, tabs, or other specific delimiters.

fill To place as much text on a line as will fit, regardless of how the text occurs in the input file.

floating keep Begins with `.KF` and ends with `.KE`. If the number of lines in a block of text exceeds the remaining lines on the page and it is necessary to force a page break, the regular text material continues to print until it reaches the end of the page, and the block of text is printed. It differs from a static keep in that it waits for a natural page break rather than forcing one.

font A collection of letters and characters unified by a distinctive pattern or "look." Times Roman, for example, is the default font for `troff`.

footer A line of text that is printed on the bottom of every page.

formatter A utility that processes text for output to a device. The `nroff` and `troff` utilities, for example, are formatters that justify the margins, center the titles, number the pages, and perform other enhancements that improve the printed appearance of text files.

grap A preprocessor for `pic` that permits inclusion of graphs in a document formatted with `troff`. Specifications for the graph are enclosed within `.G1` and `.G2` pairs and are translated by `grap` into `pic` code.

header A line of text that is printed on the top of every page.

leader A single character, repeated as necessary, to visually tie one item to another in a text line. For example, a heading and page number in a table of contents are often connected with a line of dots. The leader character is a period by default and may be set using the `.lc troff` request.

ligature Two or more characters or letters linked together. Two ligatures are available in the `troff` character set: (`fi` and (`fl`. They may be input (even in the `nroff` formatter) by \(`fi` and \(`fl`, respectively. Note that ligature mode is normally on in the `troff` formatter; that is, ligatures are automatically produced.

list-end macro Identifies the end of a list. It terminates the list and restores the previous indentation.

list-initialization macro Determines the style of the list: line spacing, indentation, marking with special symbols, and numbering or alphabetizing of list items.

list-item macro Identifies unique items to the system. It is followed by the actual text of the corresponding list items.

local motion Vertical and horizontal motion contained within a line. The function $\v'n'$ is used for vertical motion, and the function $\h'n'$ is used for horizontal motion. The distance n may be negative; the positive directions are rightward and downward. 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 0.

macro A collection of instructions or requests invoked by a single, simplified command. Text processing macros, for example, are embedded in a file and usually take the form *.XX*, where *X* is generally a capital letter. Each macro is an abbreviation for a collection of requests that would otherwise require repetition.

macro package A collection of macros grouped into a useful unit.

mm General purpose text-formatting macros for use with nroff and troff.

neqn An nroff preprocessor that formats mathematical symbols and equations using standard keyboard symbols to approximate the mathematical symbols requested as closely as possible.

nroff A formatter that produces typewriter-quality output.

number register Where troff keeps track of many of the parameters governing the page layout. You can create a number register with the command .nr or change existing parameters, such as .nr Si 8, which changes the standard indent for displays.

output device Typically, a printer or display device, such as a digital typesetter or phototypesetter, laser-driven printer, high-resolution video display terminal, terminal screen, dot matrix printer, or daisy wheel printer.

output translation A process by which one character can be made a stand-in for another character using the .tr request.

page footer Text printed at the bottom of each page.

page header Text printed at the top of each page.

page offset The distance between the left margin and the left edge of the paper. The default page offset for nroff/troff is one inch.

pic A troff preprocessor that produces simple line drawings in a document. The basic figures are arrow, box, circle, line, arc, ellipse, and text. Descriptions are included between .PS and .PE pairs.

pica A measurement (1/6 inch or 6 points) used for specifying line lengths and page lengths.

point Used to specify size of type using printer's measurement of a point equal to 1/72 of an inch. The default point size for troff is 10-point type.

preprocessor A utility such as tbl or eqn used to translate your input into commands that troff can

understand before piping the output to that process. Also the part of a compiler that provides file inclusion and macro substitution.

request Built-in command recognized by the formatters.

static keep A mechanism for preserving the integrity of a block of text. Begins with .KS and ends with .KE. If the number of lines within these two macros exceeds the remaining lines on the page, a page break is forced, and the material in the block is printed on the next page.

string A named group of characters, not including a newline character, that may be interpolated by name at any point.

tab leader A string of repeated characters between a tab stop and the next tab or end of line. A column entry followed by \a will repeat the leader character to the next entry. The default leader character is a period. A different character can be specified with the .lc instruction.

tbl A text preprocessor to troff that formats tables. Table specifications and text are placed between the commands .TS and .TE. Columns can be centered, right adjusted, left adjusted, or aligned by decimal points. Headings may be placed over single columns or groups of columns. Any table or element can be enclosed in a box, and vertical and horizontal lines can be placed at will.

text line A line destined to be printed or displayed.

transparent throughput An input line beginning with a \! that 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 postprocessor or to embed control lines in a macro created by a diversion.

trap A mechanism used in writing macros to interrupt processing in order to divert to another routine appropriate for the situation. Three types of trap mechanisms are available: page traps, diversion traps, and input-line-count traps.

troff A formatter that produces high-quality output on a high-resolution typesetter or laser printer.

unpaddable space A space that cannot be expanded during justification.

vertical spacing The vertical distance from the base line of one line of text to the base line of the next.

width function The width function $\backslash w$ '*string*' generates the numeric width of *string* (in basic units). Size and font changes may be embedded in *string* and will not affect the current environment.

word A string of characters bounded at each end by one or more of the following: the space character, the tab character, the beginning of the input line, or the end of the input line.

Index

The Apple Publishing System

A/UX Text-Processing Tools was written, edited, and composed on a desktop publishing system using Apple Macintosh computers and Microsoft Word software. Proof pages were created on Apple LaserWriter printers. Final pages were created on the Varityper VT600. Line art was created using Adobe Illustrator. PostScript®, the page-description language for the LaserWriter, was developed by Adobe Systems Incorporated.

Text type and display type are Apple's corporate font, a condensed version of ITC Garamond. Bullets are ITC Zapf Dingbats®. Some elements, such as program listings, are set in Apple Courier.