ALTOS

UNIX™ System V DOCUMENTER'S WORKBENCH™ Volume Two

Macro Package and Preprocessor Reference

UNIX™ System V DOCUMENTER'S WORKBENCH™

Volume Two

ACKNOWLEDGEMENTS

The Altos logo, as it appears in this manual, is a registered trademark of Altos Computer Systems.

DOCUMENTER'S WORKBENCH™ is a trademark of AT&T Technologies.

IMPRINT® and IMAGEN® are registered trademarks of the IMAGEN Corporation.

TEKTRONIX® is a registered trademark of Tektronics, Inc.

TELETYPE™ is a trademark of AT&T Teletype Corporation.

TRENDATA® is a registered trademark of Trendata Corporation.

UNIX™ is a trademark of AT&T Bell Laboratories.

CONTENTS

Chapter 1

INTRODUCTION

Chapter 1

INTRODUCTION

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

- *Introduction and Reference Manual*
- *Text Formatters Reference*
- *Preprocessor Reference*

Each of the chapters in this book is a user guide to a specific macro package. Information is provided in each chapter that will allow the user to understand and use the macros and access information quickly. The beginning user should refer to the *Introduction and Reference Manual* for a better overall description of the text processing tools available.

What is a Macro?

On the UNIX system, the text formatting programs (**nroff**, **troff**, **otroff**, and **sroff**) provide control of text format by the use of requests (sometimes called formatter primitives) that are mixed in with the text to be formatted. These requests normally consist of two lowercase letters preceded by a period, on a line by themselves in the text file. The request may be followed on the same line by numbers or letters that provide the formatter with more information

---

* Trademark of AT&T Bell Laboratories.

† Trademark of AT&T Technologies.

about the function of the request. The formatter requests provide a low-level control of text formatting for items such as indention, line length, spacing, filling, adjusting, centering, and titles.

One of the most useful functions of the formatters is the ability to define a group of formatter requests into a single request called a macro. The macro is given a one- or two-character name and is called, using the macro name, in the same manner as a formatter request. The normal call to a macro consists of a period at the beginning of a line, followed immediately by the one- or two-character macro name and any arguments. The arguments may consist of letters or numbers and each argument must be separated from the previous argument by a space. The arguments may be used inside the macro by the formatter requests. This is called argument substitution and it allows the user to provide specific information for the macro each time it is called. This process provides great flexibility in the function of a macro.

Macros allow the user to define powerful text formatting functions that can be called by a single name and modified easily. The use of macros simplifies the task of formatting a document. For more information on defining macros, refer to the book *Text Formatters Reference.*

Macro Packages

Several predefined macro packages that can be used with the text formatters are provided in the DOCUMENTER'S WORKBENCH software. Each of these packages contain a set of macros designed to be flexible and useful for most text formatting needs. The macro packages that are covered in this book are described below.

- **Memorandum (MM) Macros**: Chapter 2—These are the standard, general-purpose macros that work with the text

formatters **nroff**, **troff**, and **otroff**. They provide all the macros usually needed to produce a wide variety of document styles ranging from a simple letter to a several hundred page book (such as this). Documents can be printed on a simple printer using **nroff** or on a phototypesetter using **troff** or **otroff**.

- **Sroff/MM Macros**: Chapter 3—These are a set of macros styled after the memorandum macros but designed to work with the **sroff** text formatter. **Sroff**/MM macros do not have the full capability of MM macros, and since they work with **sroff**, they cannot produce output for a phototypesetter. They are designed for documents that will be produced on a printer, but because of their simplicity, the time required to format a document is greatly reduced from that of MM macros with **nroff**, **troff**, or **otroff**. With a few exceptions, **sroff**/MM macros are compatible with MM macros (macro names, arguments, and functions), so a document could be produced from the same text file using either package.

- **Viewgraph (MV) Macros**: Chapter 4—These are a set of special-purpose macros designed to produce viewgraphs and slides using the **troff** or **otroff** formatter. The MV package provides several macros useful in controlling the format of text within a viewgraph or slide.

Note: The acronym "MM" is used to refer to the *Memorandum Macro* package as used by the **nroff**, **troff**, and **otroff** formatters. The term **sroff**/MM is used when the discussion is about the macro package used by the **sroff** formatter. The term **nroff**/MM is used to differentiate between the **nroff** and **sroff** formatters when discussions are about the respective macro packages. The term **mm** refers to the command, or option, used to invoke the MM macro package.

Chapter 2

MEMORANDUM MACROS USER GUIDE

Chapter 2

MEMORANDUM MACROS USER GUIDE

1. Introduction

1.1 Purpose

This chapter is a guide and reference manual for users of Memorandum Macros (MM). These macros provide a general purpose package of text formatting macros for use with the UNIX operating system text formatters **nroff** and **troff/otroff.**

1.2 Conventions

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

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

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

In those cases in which the behavior of the two formatters **nroff** and **troff** is obviously different, the **nroff** formatter output is described first with the **troff** formatter output following in parentheses.

For Example:

    The title is underlined (italic).

means that the title is underlined by the **nroff** formatter and italicized by the **troff** formatter.

1.3 UNIX System Commands

Manual pages for the UNIX system commands are documented in the UNIX system reference manuals. These UNIX system reference manuals provide a brief description of the UNIX system commands and their options and the format for using these commands. A UNIX system command will have the form **name** when used in this guide, where the term **name** refers to a command name that can be found in a UNIX system reference manual. The following list contains the commands identified in this guide. In addition, the list categorizes the commands by the reference manual in which they can be found.

1. *UNIX System User Reference Manual* — **300, 4014, 450, col, ed, hp, sh,** and **spell.**

2. *UNIX System Programmer Reference Manual* — **term** and **termio.**

3. *Introduction and Reference Manual* — **checkmm, mmlint, eqn, eqnchar, mm, mmlint, mmt, mv, mvt, neqn, nroff, ocw, otroff, sroff, tbl,** and **troff.**

1.4 Document Structure

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

- *Parameter setting segment* sets the general style and appearance of a document. The user can control page width, margin justification, numbering styles for heading and lists, page headers and footers {9}, and many other properties of the document. Also, the user can add macros or redefine existing ones. This segment can be omitted entirely if the user is satisfied with default values; it produces no actual output, but performs only the formatter setup for the rest of the document.

- *Beginning segment* includes those items that occur only once, at the beginning of a document, e.g., title, author's name, date.

- *Body segment* is the actual text of the document. It may be as small as a single paragraph or as large as hundreds of pages. It

may have a hierarchy of headings up to seven levels deep {4}. Headings are automatically numbered (if desired) and can be saved to generate the table of contents. Five additional levels of subordination are provided by a set of list macros for automatic numbering, alphabetic sequencing, and "marking" of list items {5}. The body may also contain various types of displays, tables, figures, references, and footnotes {7, 8, 11}.

- *Ending segment* contains those items that occur only once at the end of a document. Included are signature(s) and lists of notations (e.g., "Copy to" lists) {6.11}. Certain macros may be invoked here to print information that is wholly or partially derived from the rest of the document, such as the table of contents or the cover sheet for a document {10}.

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

1.5 Input Text Structure

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

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

1.6 Definitions

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

> ***Note:*** Throughout this chapter, a reference to **troff** also means **otroff** unless otherwise indicated.

Requests are built-in commands recognized by the formatters. Although a user seldom needs to use these requests directly {3.10}, this section {1} contains references to some of the requests. For example, the request

```
.sp
```

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

Macros are named collections of requests. Each macro is an abbreviation for a collection of requests that would otherwise require repetition. The MM package supplies many macros, and the user can define additional ones. Macros and requests share the same set of names and are used in the same way.

A complete listing of memorandum macros is given in the *MM Macro Name Summary* {16} section of this chapter.

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

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

may result in the following output:

```
Today is July 25, 1983.
```

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

```
.ds DT 01/01/79
```

by invoking a macro designed for that purpose {6.8}. A listing of MM string names is given in the *MM String Name Summary* {17} section of this chapter.

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

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

A complete listing of MM number registers is given in the *MM Number Register Summary* {18} section of this chapter.

Paragraph 14.1 contains naming conventions for requests, macros, strings, and number registers. The last four sections of this chapter list all macros, strings, number registers, and error messages defined in MM.

2. Usage

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

2.1 The *mm* Command

The **mm** command can be used to prepare documents using the **nroff** formatter and the MM macro package; this command invokes **nroff** with the -cm flag {2.2}. The **mm** command has options to specify preprocessing by **tbl** and/or by **neqn** and for postprocessing by various output filters.

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

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

OPTION	*MEANING*
-e	The **neqn** is to be invoked; also causes **neqn** to read */usr/pub/eqnchar* (see **eqnchar**).
-t	The **tbl** preprocessor is to be invoked.
-c	The **col** postprocessor is to be invoked.
-E	The -e option of the **nroff** formatter is to be invoked.
-y	The uncompacted macros (-mm) are to be used instead of compacted macros (-cm).
-12	The 12-pitch mode is to be used. The pitch switch on the terminal should be set to 12 if necessary.
-T450	Output is to a DASI 450. This is the default terminal type (unless $TERM is set; see **sh**). It is also equivalent to -T1620.

-T450-12	Output is to a DASI 450 in 12-pitch mode.
-T300	Output is to a DASI 300 terminal.
-T300-12	Output is to a DASI 300 in 12-pitch mode.
-T300s	Output is to a DASI 300S.
-T300s-12	Output is to a DASI 300S in 12-pitch mode.
-T4014	Output is to a TEKTRONIX* 4014.
-T37	Output is to a TELETYPE† Model 37.
-T382	Output is to a DTC-382.
-T4000a	Output is to a TRENDATA‡ 4000A.
-TX	Output is prepared for an EBCDIC line printer.
-Thp	Output is to a Hewlett-Packard 262x or 264x (implies -c).
-T43	Output is to a TELETYPE Model 43 (implies -c).
-T40/4	Output is to a TELETYPE Model 40/4 (implies -c).
-T745	Output is to a Texas Instrument 700 series terminal (implies -c).
-T2631	Output is prepared for a Hewlett-Packard 2631 printer where -T2631-e and -T2631-c may be used for expanded and compressed modes, respectively (implies -c).
-Tlp	Output is to a device with no reverse or partial line motions or other special features (implies -c).

* Registered Trademark of Tektronix, Inc.

† Trademark of AT&T Teletype Corporation

‡ Registered Trademark of Trendata Corporation

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**).

2.2 The –cm or –mm Flag

The MM package can also be invoked by including the -cm or -mm flag as an argument to the formatter. The -cm flag causes the precompacted version of the macros to be loaded.

> ***Note:*** The -cm option cannot be used with **troff** (device independent). The **troff** formatter does not allow compacted macros. The -cm option can be used with **otroff** (old troff).

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.

2.3 Typical Command Lines

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

- Text without tables or equations:

```
mm [options] file ...
or
nroff [options] -cm  file ...

mmt [options] file ...
or
troff [options] -mm  file ...
or
otroff [options] -cm  file ...
```

- Text with tables:

```
mm -t  [options] file ...
or
tbl file ... | nroff [options] -cm

mmt -t  [options] file ...
or
tbl file ... | troff [options] -mm
or
tbl file ... | otroff [options] -cm
```

- Text with equations:

```
mm -e  [options] file ...
or
neqn /usr/pub/eqnchar file ... | nroff [options] -cm

mmt -e  [options] file ...
or
eqn /usr/pub/eqnchar file ... | troff [options] -mm
or
eqn /usr/pub/eqnchar file ... | otroff [options] -cm
```

- Text with both tables and equations:

```
mm -t -e [options] file ...
or
tbl file ... | neqn /usr/pub/eqnchar - | nroff [options] -cm

mmt -t -e [options] file ...
or
tbl file ... | eqn /usr/pub/eqnchar - | troff [options] -mm
or
tbl file ... | eqn /usr/pub/eqnchar - | otroff [options] -cm
```

Note: On any line shown above with a call to **otroff** (or **nroff**) using the -cm option, the -mm option may be used instead of -cm.

When formatting a document with the **nroff** processor, the output should normally be processed for a specific type of terminal. This is because the output may require some features that are specific to a given terminal, e.g., reverse paper motion or half-line paper motion in both directions.

Some commonly used terminal types and the command lines appropriate for them are given below. More information is found in paragraph 2.4 of this part, and **300**, **450**, **4014**, **hp**, **col**, **termio**, and **term** of the *UNIX System User Reference Manual.*

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

```
mm file ...
or
nroff -T450 -h -cm file ...
```

- 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 file ...
or
nroff -T450-12 -h -cm file ...
```

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

```
mm -12 -rW80 -rO3 file ...
or
nroff -T450-12 -rW80 -rO3 -h -cm file ...
```

- Hewlett-Packard HP262x or HP264x CRT family:

```
mm -Thp file ...
or
nroff -cm file ... | col | hp
```

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

```
mm -T745 file ...
or
nroff -cm file ... | col -x
```

The **tbl** and **eqn/neqn** preprocessors must be invoked as shown in the command lines illustrated earlier.

If 2-column processing {12.4} is used with the **nroff** formatter, either the -c option must be specified to **mm** [**mm** uses **col** automatically for many terminal types {2.1}] or the **nroff** formatter output must be postprocessed by **col**. 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** must be processed by the appropriate terminal filter (e.g., **450**); **mm** with the -c option handles all this automatically.

2.4 Parameters Set From Command Line

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

-rA*n* — *n* = 1 has effect of invoking the .AF macro without an argument {6.9}.

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

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

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

-rL*k* — sets length of physical page to *k* lines.
For the **nroff** formatter, *k* is an unscaled number representing lines.
For the **troff** formatter, *k* must be scaled.
Default value is 66 lines per page.

This flag is used, for example, when directing output to a Versatec* printer.

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

n	*PAGE 1*	*PAGES 2-END*
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

Figure 2-1. Table Showing Effects of The N Register on Page Numbering Style

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

* Registered Trademark of Versatec Corporation.

-rO*k* offsets output *k* spaces to the right.
For the **nroff** formatter, *k* is an unscaled number representing lines or character positions.
For the **troff** formatter, *k* must be scaled.
This flag is helpful for adjusting output positioning on some terminals. The default offset if this register is not set on the command line is 0.75 inch (**nroff**) and 0.5 inch (**troff**).

Note: This register name is the capital letter "O".

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

-rS*n* sets point size and vertical spacing for the document.
The default *n* is 10, i.e., 10-point type on 12-point vertical spacing, giving six lines per inch {12.10}.
This flag applies to the **troff** formatter only.

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

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

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

2.5 Omission of *-cm* or *-mm* Flag

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

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

In this case, the user must not use the -cm or -mm flag [nor the **mm** or **mmt** command]; the **.so** request has the equivalent effect, but registers shown in paragraph 2.4 must be initialized before the **.so** request because their values are meaningful only if set before macro definitions are processed. When using this method, it is best to lock into the input file only those parameters that are seldom changed. For example:

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

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

3. Formatting Concepts

3.1 Basic Terms

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

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

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

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

3.2 Arguments and Double Quotes

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

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

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

3.3 Unpaddable Spaces

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

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

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

```
.tr ~
```

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

```
.tr ~~
```

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

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

3.4 Hyphenation

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

```
.nr Hy 1
```

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

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

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

The user may specify a different hyphenation indicator.

```
.HC [hyphenation-indicator]
```

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

```
.HC ^
```

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

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

```
.hw print-out
```

3.5 Tabs

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

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

```
.ta 9 17 25 33 41 49 57 ... 161
```

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

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

3.6 BEL Character

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

3.7 Bullets

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

```
\*(BU
```

The bullet list (.BL) macro {5.1.1.2} uses this string to generate automatically the bullets for bullet listed items.

3.8 Dashes, Minus Signs, and Hyphens

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

- Users who intend to use the **nroff** formatter may use only the minus sign (−) for the minus, hyphen, and dash.
- Users who plan to use the **troff** formatter primarily should follow **troff** escape conventions.

- Users who plan to use both formatters must take care during input text file preparation. Unfortunately, these graphic characters cannot be represented in a way that is both compatible and convenient for both formatters.

The following approach is suggested:

Dash — Type "*(EM" for each text dash for both **nroff** and **troff** formatters. This string generates an em dash in the **troff** formatter and two dashes (--) in the **nroff** formatter. Dash list (.DL) macros {5.1.1.3} automatically generate the em dash for each list item.

Hyphen — Type "-" and use as is for both formatters. The **nroff** formatter will print it as is. The **troff** formatter will print - (a true hyphen).

Minus — Type "\-" for a true minus sign regardless of formatter. The **nroff** formatter will ignore the \. The **troff** formatter will print a true minus sign.

3.9 Trademark String

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

For example:

```
The
.I
UNIX
.R
\*(Tm
.I
System User Reference Manual
.R
is available from the library.
```

yields:

> The *UNIX* ™ *System User Reference Manual* is available from the library.

3.10 Use of Formatter Requests

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

.af	Assign format
.br	Break
.ce	Center
.de	Define macro
.ds	Define string
.fi	Fill output lines
.hw	Exception word
.ls	Line spacing
.nf	No filling of output lines
.nr	Define and set number register
.nx	Go to next file (does not return)
.rm	Remove macro
.rr	Remove register
.rs	Restore spacing
.so	Switch to source file and return
.sp	Space
.ta	Tab stop settings
.ti	Temporary indent
.tl	Title
.tr	Translate
.!	Escape

The **.fp**, **.lg**, and **.ss** requests are also sometimes useful for the **troff** formatter. Use of other requests without fully understanding their implications very often leads to disaster.

4. Paragraphs and Headings

4.1 Paragraphs

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

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

4.1.1 Paragraph Indention

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

type	*RESULT*
0	left justified
1	indent

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

.nr Pi 10

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

- The initial value of *Pt* is 0, which provides left-justified paragraphs.

- All paragraphs can be forced to be indented by inserting the following at the beginning of the document input file:

```
.nr Pt 1
```

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

```
.nr Pt 2
```

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

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

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

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

4.1.2 Numbered Paragraphs

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

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

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

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

4.1.3 Spacing Between Paragraphs

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

4.2 Numbered Headings

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

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

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

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

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

4.2.1 Normal Appearance

The effect of the .H macro varies according to the *level* argument. First-level headings are preceded by two blank lines (one vertical space); all others are preceded by one blank line (one-half a vertical space). The following table describes the default effect of the *level* argument.

.H 1 heading-text	Produces an underlined (italicized) font heading followed by a single blank line (one-half a vertical space). The following text begins on a new line and is indented according to the current paragraph type. Full capital letters should be used to make the heading stand out.
.H 2 heading-text	Produces an underlined (italicized) font heading followed by a single blank line (one-half a vertical space). The following text begins on a new line and is indented according to the current paragraph type. Initial capitals should be used in the heading text.
.H *n* heading-text	Produces an underlined (italicized) heading followed by two spaces ($3 \leq n \leq 7$). The following text begins on the same line.

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

The following listing gives the first few .H calls used for this part:

```
.H 1 " Paragraphs and Headings"
.H 2 " Paragraphs"
.H 3 " Paragraph Indention"
.H 3 " Numbered Paragraphs"
.H 3 " Spacing Between Paragraphs"
.H 2 " Numbered Headings"
.H 3 " Normal Appearance"
.H 3 " Altering Appearance"
.H 4 " Prespacing and Page Ejection"
.H 4 " Spacing After Headings"
.H 4 " Centered Headings"
.H 4 " Bold, Italic, and Underlined Headings"
.H 5 " Control by Level:"
```

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

4.2.2 Altering Appearance

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

4.2.2.1 Prespacing and Page Ejection

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

```
.nr Ej 1
```

at the beginning of the document input text file. Long documents may be made more manageable if each section starts on a new page. Setting the *Ej* register to a higher value causes the same effect for

headings up to that level, i.e., a page eject occurs if the heading level is less than or equal to the *Ej* value.

4.2.2.2 Spacing After Headings

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

- If the heading level is less than or equal to *Hb*, a break {3.1} occurs after the heading.
- If the heading level is less than or equal to *Hs*, a blank line (one-half a vertical space) is inserted after the heading.
- If a heading level is greater than *Hb* and also greater than *Hs*, then the heading (if any) is immediately followed by text on the same line.

These registers permit headings to be separated from the text in a consistent way throughout a document while allowing easy alteration of white space and heading emphasis. The default value for *Hb* and *Hs* is 2.

For any stand-alone heading, i.e., a heading on a line by itself, alignment of the next line of output is controlled by the *Hi* number register.

- If *Hi* is 0, text is left-justified.
- If *Hi* is 1 (the default value), text is indented according to the paragraph type as specified by the *Pt* register {4.1}.
- 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 (one-half a vertical space) to appear after the first three heading levels, to have no run-in headings, and to force the text following all headings to be left-justified (regardless of the value of *Pt*), the following should appear at the beginning of the document input text file:

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

4.2.2.3 Centered Headings

The *Hc* register can be used to obtain centered headings. A heading is centered if its *level* argument is less than or equal to *Hc* and if it is also a stand-alone heading {4.2.2.2}. The *Hc* register is 0 initially (no centered headings).

4.2.2.4 Bold, Italic, and Underlined Headings

4.2.2.4.1 Control by Level: Any heading that is underlined by the **nroff** formatter is italicized by the **troff** formatter. The string *HF* (heading font) contains seven codes that specify fonts for heading levels 1 through 7. Legal codes, code interpretations, and defaults for *HF* codes are shown in Figure 2-2.

FORMATTER	*HF CODE*			*DEFAULT HF CODE*
	1	*2*	*3*	
nroff	no underline	underline	bold	2 2 2 2 2 2 2
troff	Roman	italic	bold	2 2 2 2 2 2 2

Figure 2-2. Table Of HF String Codes, Effects, and Default Values

Thus, all levels 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 five bold levels and two underlined (italic) levels:

.ds HF 3 3 3 3 3

4.2.2.4.2 NROFF Underlining Style: The **nroff** formatter underlines in either of two styles:

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

By default, MM attempts to use the continuous style on any heading that is to be underlined and is short enough to fit on a single line. If a heading is to be underlined but is longer than a single line, the heading is underlined in the normal style.

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

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

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

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

.ds HP 12 12 10 10 10 10 10

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

.ds HP +2 +2 -1 -1

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

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

> ***Note:*** Only the point size of the headings is affected. Specifying a large point size without providing increased vertical spacing (via .HX and/or .HZ {4.6}) may cause overprinting.

4.2.2.5 Marking Styles—Numerals and Concatenation

.HM [arg1] ... [arg7]

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

a string indicating the type of marking to be used. Legal arguments and their meanings are:

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

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

For example, a commonly used "outline" style is obtained by:

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

4.3 Unnumbered Headings

.HU heading-text

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

.HU heading-text

and

.H 2 heading-text

is the printing of the heading mark for the latter. Both macros have the effect of incrementing the numbering counter for level 2 and resetting to zero the counters for levels 3 through 7. Typically, the value of *Hu* should be set to make unnumbered headings (if any) be the lowest-level headings in a document.

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

4.4 Headings and Table of Contents

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

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

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

Due to the way headings are saved, it is possible to exceed the formatter's storage capacity, particularly when saving many levels of many headings, while also processing displays {7} and footnotes {8}. If this happens, the "Out of temp file space" formatter error message {19.2} will be issued; the only remedy is to save fewer levels and/or to have fewer words in the heading text.

4.5 First-Level Headings and Page Numbering Style

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

4.6 User Exit Macros

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

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

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

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

By the time .H calls .HX, it has already incremented the heading counter of the specified level {4.2.2.5}, produced blank lines (vertical spaces) to precede the heading {4.2.2.1}, and accumulated the "heading mark", i.e., the string of digits, letters, and periods needed for a numbered heading. When .HX is called, all user-accessible registers and strings can be referenced, as well as the following:

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

string.
If *rlevel* is 0, this string is null.

register ;0 — This register indicates the type of spacing that is to follow the heading {4.2.2.2}.
A value of 0 means that the heading is run-in.
A value of 1 means a break (but no blank line) is to follow the heading.
A value of 2 means that a blank line (one-half a vertical space) 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 1 otherwise. The **.ne** request is for the following number of lines: the contents of the "register ;0" taken as blank lines (halves of vertical space) plus the contents of "register ;3" as blank lines (halves of vertical space) plus the number of lines of the heading.

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

- Change first-level heading mark from format *n.* to *n.*0:
  .if \\$1=1 .ds }0 \\n(H1.0\<sp>\<sp>
  (where <sp> stands for a space)

- Separate run-in heading from the text with a period and two unpaddable spaces:
  .if \\n(;0=0 .ds }2 .\<sp>\<sp>

- Assure that at least 15 lines are left on the page before printing a first-level heading:
  .if \\$1=1 .nr ;3 (15-\\n(;0)v

- Add three additional blank lines before each first-level heading:
  .if \\$1=1 .sp 3

- Indent level 3 run-in headings by five spaces:
  .if \\$1=3 .ti 5n

If temporary strings or macros are used within .HX, their names should be chosen with care {14.1}.

When the .HY macro is called after the .ne is issued, certain features requested in .HX must be repeated.

For example:

```
.de HY
.if \\$1=3 .ti 5n
..
```

The .HZ macro is called at the end of .H to permit user-controlled actions after the heading is produced. In a large document, sections may correspond to chapters of a book; and the user may want to change a page header or footer, e.g.:

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

4.7 Hints for Large Documents

A large document is often organized for convenience into one input text file per section. If the files are numbered, it is wise to use enough digits in the names of these files for the maximum number of sections, i.e., use suffix numbers 01 through 20 rather than 1 through 9 and 10 through 20.

Users often want to format individual sections of long documents. To do this with the correct section numbers, it is necessary to set register *H1* to one less than the number of the section just before the corresponding **.H 1** call. For example, at the beginning of Part 5, insert

```
.nr H1 4
```

Caution: This is not good practice. It defeats the automatic (re)numbering of sections when sections are added or deleted. Such lines should be removed as soon as possible.

5. Lists

This part describes different styles of lists; automatically numbered and alphabetized lists, bullet lists, dash lists, lists with arbitrary marks, and lists starting with arbitrary strings, i.e., with terms or phrases to be defined.

5.1 List Macros

In order to avoid repetitive typing of arguments to describe the style or appearance of items in a list, MM provides a convenient way to specify lists. All lists share the same overall structure and are composed of the following basic parts:

- A *list-initialization macro* (.AL .BL, .DL, .ML, .RL, or .VL) determines the style of list: line spacing, indentation, marking with special symbols, and numbering or alphabetizing of list items {5.1.1}.

- One or more *list-item macros* (.LI) identifies each unique item to the system. It is followed by the actual text of the corresponding list item {5.1.2}.

- The *list-end macro* (.LE) identifies the end of the list. It terminates the list and restores the previous indentation {5.1.3}.

Lists may be nested up to six levels. The list-initialization macro saves the previous list status (indentation marking style, etc.); the .LE macro restores it.

With this approach, the format of a list is specified only once at the beginning of the list. In addition, by building onto the existing structure, users may create their own customized sets of list macros with relatively little effort ({5.2} and {5.3}).

5.1.1 List-Initialization Macros

List-initialization macros are implemented as calls to the more basic .LB macro {5.2}. They are:

.AL	Automatically Numbered or Alphabetized List {5.1.1.1}
.BL	Bullet List {5.1.1.2}
.DL	Dash List {5.1.1.3}
.ML	Marked List {5.1.1.4}
.RL	Reference List {5.1.1.5}
.VL	Variable-Item List {5.1.1.6}

5.1.1.1 Automatically Numbered or Alphabetized List

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

The **.AL** macro is used to begin sequentially numbered or alphabetized lists. If there are no arguments, the list is numbered; and text is indented by *Li* (initially six) spaces from the indent in force when the .AL is called. This leaves room for a space, two digits, a period, and two spaces before the text. Values that specify indentation must be unscaled and are treated as "character positions", i.e., number of ens.

Spacing at the beginning of the list and between items can be suppressed by setting the list space register (*Ls*). The *Ls* register is set to the innermost list level for which spacing is done. For example:

.nr Ls 0

specifies that no spacing will occur around any list items. The default value for *Ls* is six (which is the maximum list nesting level).

- The *type* argument may be given to obtain a different type of sequencing. Its value indicates the first element in the sequence

desired. If *type* argument is omitted or null, the value 1 is assumed.

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

- If *text-indent* argument is non-null, it is used as the number of spaces from the current indent to the text; i.e., it is used instead of the *Li* register for this list only. If *text-indent* argument is null, the value of *Li* will be used.

- If the third argument is given, a blank line (one-half a vertical space) will not separate items in the list. A blank line will occur before the first item however.

5.1.1.2 Bullet List

.BL [text-indent] [1]

The **.BL** macro begins a bullet list. Each list item is marked by a bullet (•) followed by one space.

- If the *text-indent* argument is non-null, it overrides the default indentation (the amount of paragraph indentation as given in the *Pi* register {4.1.1}). In the default case, the text of a bullet list lines up with the first line of indented paragraphs.

- If the second argument is specified, no blank lines will separate items in the list.

5.1.1.3 Dash List

.DL [text-indent] [1]

The **.DL** macro begins a dash list. Each list item is marked by a dash (—) followed by one space.

- If the *text-indent* argument is non-null, it overrides the default indentation (the amount of paragraph indentation as given in the *Pi* register {4.1.1}). In the default case, the text of a dash list lines up with the first line of indented paragraphs.
- If the second argument is specified, no blank lines will separate items in the list.

5.1.1.4 Marked List

.ML mark [text-indent] [1]

The **.ML** macro is much like .BL and .DL macros but expects the user to specify an arbitrary *mark* which may consist of more than a single character.

- Text is indented *text-indent* spaces if the second argument is not null; otherwise, the text is indented one more space than the width of *mark*.
- If the third argument is specified, no blank lines will separate items in the list.

Note: The *mark* must not contain ordinary (paddable) spaces because alignment of items will be lost if the right margin is justified {3.3}.

5.1.1.5 Reference List

.RL [text-indent] [1]

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

- If *text-indent* argument is non-null, it is used as the number of spaces from the current indent to the text; i.e., it is used instead of *Li* for this list only. If *text-indent* argument is omitted or null, the value of *Li* is used.
- If the second argument is specified, no blank lines will separate the items in the list.

5.1.1.6 Variable-Item List

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

When a list begins with a **.VL** macro, there is effectively no current *mark*; it is expected that each .LI will provide its own mark. This form is typically used to display definitions of terms or phrases.

- *Text-indent* provides the distance from current indent to beginning of the text.
- *Mark indent* produces the number of spaces from current indent to beginning of the *mark*, and it defaults to 0 if omitted or null.
- If the third argument is specified, no blank lines will separate items in the list.

An example of .VL macro usage is shown below:

```
.tr ~
.VL 20 2
.LI mark~1
Here is a description of mark 1;
"mark 1" of the .LI line contains a tilde
translated to an unpaddable space in order
to avoid extra spaces between
"mark" and "1" {3.3}.
.LI second~mark
This is the second mark also using a tilde translated
to an unpaddable space.
.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 effectively has no mark because the
tilde following the .LI is translated into a space.
.LE
```

when formatted yields:

mark 1
: Here is a description of mark 1; "mark 1" of the .LI line contains a tilde translated to an unpaddable space in order to avoid extra spaces between "mark" and "1" {3.3}.

second mark
: This is the second mark also using a tilde translated to an unpaddable space.

third mark longer than indent:
: This item shows the effect of a long mark; one space separates the mark from the text.

  This item effectively has no mark because the tilde following the .LI is translated into a space.

The tilde argument on the last .LI above is required; otherwise, a "hanging indent" would have been produced. A "hanging indent" is produced by using .VL and calling .LI with no arguments or with a null first argument. For example:

```
.VL 10
.LI
Here is some text to show a hanging indent.
The first line of text is at the left margin.
The second is indented 10 spaces.
.LE
```

when formatted yields:

Here is some text to show a hanging indent. The first line of text is at the left margin. The second is indented 10 spaces.

Note: The *mark* must not contain ordinary (paddable) spaces because alignment of items will be lost if the right margin is justified {3.3}.

5.1.2 List-Item Macro

```
.LI [mark] [1]
one or more lines of text that make up the list item.
```

The **.LI** macro is used with all lists and for each list item. It normally causes output of a single blank line (one-half a vertical space) before its list item although this may be suppressed.

- If no arguments are given, .LI labels the item with the current *mark* 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 6
.LI
This is a simple bullet item.
.LI +
This replaces the bullet with a "plus".
.LI + 1
This uses a "plus" as prefix to the bullet.
.LE
```

when formatted yields:

- This is a simple bullet item.

+ This replaces the bullet with a "plus".

+ • This uses a "plus" as prefix to the bullet.

Note: The *mark* must not contain ordinary (paddable) spaces because alignment of items will be lost if the right margin is justified {3.3}.

If the current *mark* (in the current list) is a null string and the first argument of .LI is omitted or null, the resulting effect is that of a "hanging indent", i.e., the first line of the following text is moved to the left starting at the same place where *mark* would have started {5.1.1.6}.

5.1.3 List-End Macro

```
.LE [1]
```

The **.LE** macro restores the state of the list to that existing just before the most recent list-initialization macro call. If the optional argument is given, the .LE outputs a blank line (one-half a vertical

space). 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, .H, or .LI macro.

The .H and .HU macros automatically clear all list information. The user may omit the .LE macros that would normally occur just before either of these macros and not receive the "LE:mismatched" error message. Such a practice is not recommended because errors will occur if the list text is separated from the heading at some later time (e.g., by insertion of text).

5.1.4 Example of Nested Lists

An example of input for the several lists and the corresponding output is shown below. The .AL and .DL macro calls {5.1.1} contained therein are examples of list-initialization macros.

Input text is:

```
.AL A
.LI
This is alphabetized list item A.
This text shows the alignment of the second line
of the item.
Notice the text indentations and alignment of left
and right margins.
.AL
.LI
This is numbered item 1.
This text shows the alignment of the second line
of the item.
The quick brown fox jumped over the lazy dog's back.
.DL
.LI
This is a dash item.
This text shows the alignment of the second line
of the item.
The quick brown fox jumped over the lazy dog's back.
.LI + 1
This is a dash item with a "plus" as prefix.
This text shows the alignment of the second line
of the item.
The quick brown fox jumped over the lazy dog's back.
.LE
.LI
This is numbered item 2.
.LE
.LI
This is another alphabetized list item B.
This text shows the alignment of the second line
of the item.
The quick brown fox jumped over the lazy dog's back.
.LE
.P
This paragraph follows a list item and is aligned with
the left margin.
A paragraph following a list resumes the normal line
length and margins.
```

The output is:

A. This is alphabetized list item A. This text shows the alignment of the second line of the item. Notice the text indentations and alignment of left and right margins.

   1. This is numbered item 1. This text shows the alignment of the second line of the item. The quick brown fox jumped over the lazy dog's back.

   — This is a dash item. This text shows the alignment of the second line of the item. The quick brown fox jumped over the lazy dog's back.

+ — This is a dash item with a "plus" as prefix. This text shows the alignment of the second line of the item. The quick brown fox jumped over the lazy dog's back.

   2. This is numbered item 2.

B. This is another alphabetized list item B. This text shows the alignment of the second line of the item. The quick brown fox jumped over the lazy dog's back.

This paragraph follows a list item and is aligned with the left margin. A paragraph following a list resumes the normal line length and margins.

5.2 List-Begin Macro and Customized Lists

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

List-initialization macros described above suffice for almost all cases. However, if necessary, the user may obtain more control over the

layout of lists by using the basic list-begin macro (**.LB**). The .LB macro is used by the other list-initialization macros. Its arguments are as follows:

- The *text-indent* argument provides the number of spaces that text is to be indented from the current indent. Normally, this value is taken from the *Li* register (for automatic lists) or from the *Pi* register (for bullet and dash lists).

- The combination of *mark-indent* and *pad* arguments determines the placement of the mark. The mark is placed within an area (called *mark area*) that starts *mark-indent* spaces to the right of the current indent and ends where the text begins (i.e., ends *text-indent* spaces to the right of the current indent). The *mark-indent* argument is typically 0.

- Within the *mark area*, the mark is left justified if the *pad* argument is 0. If *pad* is a number *n* (greater than 0) then *n* blanks are appended to the mark; the *mark-indent* value is ignored. The resulting string immediately precedes the text. The *mark* is effectively right justified *pad* spaces immediately to the left of text.

- Arguments *type* and *mark* interact to control the type of marking used. If *type* is 0, simple marking is performed using the mark character(s) found in the *mark* argument. If *type* is greater than 0, automatic numbering or alphabetizing is done; and *mark* is then interpreted as the first item in the sequence to be used for numbering or alphabetizing, i.e., it is chosen from the set (1, A, a, I, i) as in {5.1.1.1}. This is summarized below:

ARGUMENT		*RESULT*
type	***mark***	
0	omitted	hanging indent
0	*string*	*string* is the mark
>0	omitted	Arabic numbering
>0	one of: 1, A, a, I, i	automatic numbering or alphabetic sequencing

Each nonzero value of *type* from one to six selects a different way of displaying the marks. The following table shows the output appearance for each value of *type*:

VALUE	*APPEARANCE*
1	*x.*
2	*x)*
3	*(x)*
4	*[x]*
5	*<x>*
6	*{x}*

where *x* is the generated number or letter.

> ***Note:*** The *mark* must not contain ordinary (paddable) spaces because alignment of items will be lost if the right margin is justified {3.3}.

- The *LI-space* argument gives the number of blank lines (halves of a vertical space) that should be output by each .LI macro in the list. If omitted, *LI-space* defaults to 1; the value 0 can be used to obtain compact lists. If *LI-space* is greater than 0, the .LI macro issues a **.ne** request for two lines just before printing the mark.

- The *LB-space* argument is the number of blank lines (one-half a vertical space) to be output by .LB itself. If omitted *LB-space* defaults to 0.

There are three combinations of *LI-space* and *LB-space*:

- The normal case is to set *LI-space* to 1 and *LB-space* to 0 yielding one blank line 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.

- 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 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.

Paragraph 5.3 shows how to build upon the supplied list of macros to obtain other kinds of lists.

5.3 User-Defined List Structures

Note: This part is intended for users accustomed to writing formatter macros.

If a large document requires complex list structures, it is useful to define the appearance for each list level only once instead of having to define the appearance at the beginning of each list. This permits consistency of style in a large document. A generalized list-initialization macro might be defined in such a way that what the macro does depends on the list-nesting level in effect at the time the macro is called. Levels 1 through 5 of the lists to be formatted may have the following appearance:

A.

[1]

•

a)

+

The following code defines a macro (.aL) that always begins a new list and determines the type of list according to the current list level. To understand it, the user should know that the number register *:g* is used by the MM list macros to determine the current list level; it is 0 if there is no currently active list. Each call to a list-initialization macro increments *:g*, and each .LE call decrements it.

```
.\" register g is used as a local temporary to save :g
.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

A. First line.

   [1] Second line.

B. Third line.

There is another approach to lists that is similar to the .H mechanism. List-initialization, as well as the .LI and the .LE macros, are all included in a single macro. That macro (defined as .bL below) requires an argument to tell it what level of item is required; it

adjusts the list level by either beginning a new list or setting the list level back to a previous value, and then issues a .LI macro call to produce the item:

```
.de bL
.ie \\n(.$ .nr g \\$1          \" argument given, that is the level
.el .nr g \\n(:g               \" no argument, use current level
.if \\ng-\\n(:g>1 .)D          \" **ILLEGAL SKIPPING OF LEVEL
.\"                              increasing level by more than 1
.if \\ng>\\n(:g \{.aL \\ng-1    \" if g > :g, begin new list
.nr                            \" and reset g to current level
.\"                              (.aL changes g)
.if \\n(:g>\\ng .LC \\ng       \" if :g > g, prune back to correct level
.\"                              if :g = g, stay within current list
.LI                            \" in all cases, get out an item
..
```

For .bL to work, the previous definition of the .aL macro must be changed to obtain the value of *g* from its argument rather than from *:g*. Invoking .bL without arguments causes it to stay at the current list level. The .LC (List Clear) macro removes list descriptions until the level is less than or equal to that of its argument. For example, the .H macro includes the call ".LC 0". If text is to be resumed at the end of a list, insert the call ".LC 0" to clear out the lists completely. The example below illustrates the relatively small amount of input needed by this approach. The input text

```
The quick brown fox jumped over the lazy dog's back.
.bL 1
First line.
.bL 2
Second line.
.bL 1
Third line.
.bL
Fourth line.
.LC 0
Fifth line.
```

when formatted yields:

The quick brown fox jumped over the lazy dog's back.

A. First line.

[1] Second line.

B. Third line.

C. Fourth line.
Fifth line.

6. Memorandum and Released-Paper Documents

One use of MM is for the preparation of memoranda and released-paper documents (a documentation style used by Bell Laboratories) which have special requirements for the first page and for the cover sheet. Data needed (title, author, date, case numbers, etc.) is entered the same for both styles; an argument to the .MT macro indicates which style is being used.

6.1 Sequence of Beginning Macros

Macros, if present, must be given in the following order:

.ND new-date
.TL [charging-case] [filing-case]
one or more lines of text
.AF [company-name]
.AU name [initials] [loc] [dept] [ext] [room] [arg] [arg]
.AT [title] ...
.TM [number] ...
.AS [arg] [indent]
one or more lines of abstract text
.AE
.NS [arg] [1]
one or more lines of "copy to" notation
.NE
.OK [keyword] ...
.MT [type] [addressee]

The only required macros for a memorandum for file or a released-paper document are .TL, .AU, and .MT; all other macros (and their associated input lines) may be omitted if the features are not needed. Once .MT has been invoked, none of the above macros (except .NS and .NE) can be reinvoked because they are removed from the table of defined macros to save memory space.

If neither the memorandum nor released-paper style is desired, the TL, AU, TM, AE, OK, MT, ND, and AF macros should be omitted

from the input text. If these macros are omitted, the first page will have only the page header followed by the body of the document.

> ***Note:*** The macros for memorandum and released-paper documents require the postprocessor *col* when you are using the **nroff** text formatter.

6.2 Title

```
.TL [charging-case] [filing-case]
one or more lines of title text
```

Arguments to the **.TL** macro are the charging-case number(s) and filing-case number(s).

- The *charging-case* argument is the case number to which time was charged for the development of the project described in the memorandum. Multiple charging-case numbers are entered as "subarguments" by separating each from the previous with a comma and a space and enclosing the entire argument within double quotes.

- The *filing-case* argument is a number under which the memorandum is to be filed. Multiple filing case numbers are entered similarly. For example:

```
.TL " 12345, 67890"  987654321
Construction of a Table of All Even Prime Numbers
```

The title of the memorandum or released-paper document follows the .TL macro and is processed in fill mode. The .br request may be used to break the title into several lines as follows:

```
.TL 12345
First Title Line
.br
\!.br
Second Title Line
```

On output, the title appears after the word "subject" in the memorandum style and is centered and printed in bold in the released-paper document style.

If only a charging case number or only a filing case number is given, it will be separated from the title in the memorandum style by a dash and will appear on the same line as the title. If both case numbers are given and are the same, then "Charging and Filing Case" followed by the number will appear on a line following the title. If the two case numbers are different, separate lines for "Charging Case" and "File Case" will appear after the title.

6.3 Authors

.AU name [initials] [loc] [dept] [ext] [room] [arg] [arg]
.AT [title] ...

The **.AU** macro receives as arguments information that describes an author. If any argument contains blanks, that argument must be enclosed within double quotes. The first six arguments must appear in the order given. A separate .AU macro is required for each author.

The **.AT** macro is used to specify the author's title. Up to nine arguments may be given. Each will appear in the signature block for memorandum style {6.11} on a separate line following the signer's name. The .AT must immediately follow the .AU for the given author. For example:

.AU " J. J. Jones" JJJ PY 9876 5432 1Z-234
.AT Director " Materials Research Laboratory"

In the "from" portion in the memorandum style, the author's name is followed by location and department number on one line and by room number and extension number on the next line. The "x" for the extension is added automatically. Printing of the location, department number, extension number, and room number may be suppressed on the first page of a memorandum by setting the register *Au* to 0; the default value for *Au* is 1. Arguments 7 through 9 of the

.AU macro, if present, will follow this normal author information in the "from" portion, each on a separate line. These last three arguments may be used for organizational numbering schemes, etc. For example:

```
.AU " S. P. LeName" SPL IH 9988 7766 5H-444 9876-543210.01MF
```

The name, initials, location, and department are also used in the signature block. Author information in the "from" portion, as well as names and initials in the signature block will appear in the same order as the .AU macros.

Names of authors in the released-paper style are centered below the title. Following the name of the last author, "Bell Laboratories" and the location are centered. The paragraph on memorandum types {6.7} contains information regarding authors from different locations.

6.4 TM Numbers

```
.TM [number] ...
```

If the memorandum is a technical memorandum, the TM numbers are supplied via the **.TM** macro. Up to nine numbers may be specified. For example:

```
.TM 7654321 77777777
```

This macro call is ignored in the released-paper and external-letter styles {6.7}.

6.5 Abstract

```
.AS [arg] [indent]
text of abstract
.AE
```

If a memorandum has an abstract, the input is identified with the **.AS** (abstract start) and **.AE** (abstract end) delimiters. Abstracts are printed on page 1 of a document and/or on its cover sheet. There are three styles of cover sheets:

- Released paper
- Technical memorandum
- Memorandum for file {10.2} (also used for engineer's notes, memoranda for record, etc.).

Cover sheets for released papers and technical memoranda are obtained by invoking the .CS macro {10.2}.

In released-paper style (first argument of the .MT macro {6.7} is 4) and in technical memorandum style if the first argument of .AS is:

0 - Abstract will be printed on page 1 and on the cover sheet (if any).

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

In memoranda for file style and in all other documents (other than external letters) if the first argument of .AS is:

0 - Abstract will appear on page 1 and there will be no cover sheet printed.

2 - Abstract will appear only on the cover sheet which will be produced automatically (i.e., without invoking the .CS macro).

It is not possible to get either an abstract or a cover sheet with an external letter (first argument of the .MT macro is 5).

Notations such as a "copy to" list {6.11.2} are allowed on memorandum for file cover sheets; the .NS and .NE macros must

appear after the .AS 2 and .AE macros. Headings {4.2, 4.3} and displays {7} are not permitted within an abstract.

The abstract is printed with ordinary text margins; an indentation to be used for both margins can be specified as the second argument of .AS. Values that specify indentation must be unscaled and are treated as "character positions", i.e., as the number of *ens*.

6.6 Other Keywords

.OK [keyword] ...

Topical keywords should be specified on a technical memorandum cover sheet. Up to nine such keywords or keyword phrases may be specified as arguments to the **.OK** macro; if any keyword contains spaces, it must be enclosed within double quotes.

6.7 Memorandum Types

.MT [type] [addressee]

The **.MT** macro controls the format of the top part of the first page of a memorandum or of a released-paper document and the format of the cover sheets. The *type* arguments and corresponding values are:

type	*VALUE*
""	no memorandum type printed
0	no memorandum type printed
none	MEMORANDUM FOR FILE
1	MEMORANDUM FOR FILE
2	PROGRAMMER'S NOTES
3	ENGINEER'S NOTES
4	released-paper style
5	external-letter style
" *string*"	*string* (enclosed in quotes)

If the *type* argument indicates a memorandum style document, the corresponding statement indicated under "VALUE" will be printed

after the last line of author information. If *type* is longer than one character, then the string, itself, will be printed. For example:

.MT " Technical Note #5"

A simple letter is produced by calling .MT with a null (but not omitted) or 0 argument.

The second argument to .MT is the name of the addressee of a letter. If present, that name and the page number replace the normal page header on the second and following pages of a letter. For example:

.MT 1 " John Jones"

produces

John Jones - 2

The *addressee* argument may not be used if the first argument is 4 (released-paper style document).

The released-paper style is obtained by specifying

.MT 4 [1]

This results in a centered, bold title followed by centered names of authors. The location of the last author is used as the location following "Bell Laboratories" (unless the .AF macro specifies a different company). If the optional second argument to .MT 4 is given, then the name of each author is followed by the respective company name and location. Information necessary for the memorandum style document but not for the released-paper style document is ignored.

If the released-paper style document is utilized, most Bell Laboratories (BTL) location codes are defined as strings that are the addresses of the corresponding BTL locations. These codes are

needed only until the .MT macro is invoked. Thus, following the .MT macro, the user may reuse these string names. In addition, the macros for the end of a memorandum {6.11} and their associated lines of input are ignored when the released-paper style is specified.

Authors from non-BTL locations may include their affiliations in the released-paper style by specifying the appropriate .AF macro {6.9} and defining a string (with a 2-character name such as ZZ) for the address before each .AU. For example:

```
.TL
A Learned Treatise
.AF " Getem Inc."
.ds ZZ " 22 Maple Avenue, Sometown 09999"
.AU " F. Swatter" "" ZZ
.AF " Bell Laboratories"
.AU " Sam P. LeName" "" CB
.MT 4 1
```

In the external-letter style document (.MT 5), only the title (without the word "subject:") and the date are printed in the upper left and right corners, respectively, on the first page. It is expected that preprinted stationery will be used with the company logo and address of the author.

6.8 Date Changes

```
.ND new-date
```

The **.ND** macro alters the value of the string *DT*, which is initially set to produce the current date. If the argument contains spaces, it must be enclosed within double quotes.

6.9 Alternate First-Page Format

.AF [company-name]

An alternate first-page format can be specified with the **.AF** macro. The words "subject", "date", and "from" (in the memorandum style) are omitted and an alternate company name is used.

If an argument is given, it replaces "Bell Laboratories" without affecting other headings. If the argument is null, "Bell Laboratories" is suppressed; and extra blank lines are inserted to allow room for stamping the document with a logo or a Bell Laboratories stamp.

The .AF with no argument suppresses "Bell Laboratories" and the "Subject/Date/From" headings, thus allowing output on preprinted stationery. The use of .AF with no arguments is equivalent to the use of -rA1 {2.4}, except that the latter must be used if it is necessary to change the line length and/or page offset (which default to 5.8i and 1i, respectively, for preprinted forms). The command line options -rOk and -rWk {2.4} are not effective with .AF. The only .AF use appropriate for the **troff** formatter is to specify a replacement for "Bell Laboratories".

The command line option -rE*n* {2.4} controls the font of the "Subject/Date/From" block.

6.10 Example

Input text for a document may begin as follows:

```
.TL
MM\*(EMMemorandum Macros
.AU " D. W. Smith" DWS PY
.AU " J. R. Mashey" JRM PY
.AU " E. C. Pariser (January 1980 Revision)" ECP PY
.AU " N. W. Smith (June 1980 Revision)" NWS PY
.MT 4
```

Figures 2-3, 2-4, and 2-5 show the input text file and both the **nroff** and **troff** formatter outputs for a simple letter.

```
.ND "May 31, 1983"
.TL 334455
Out-of-Hours Course Description
.AU "D. W. Stevenson" DWS PY 9876 5432 1X-123
.AF "Your Company"
.MT 0
.DS
J. M. Jones:
.DE
.P
Please use the following description for the out-of-hours course
.I
Document Preparation on the UNIX*
.R
.FS *
Trademark of Bell Laboratories.
.FE
.I "System:"
.P
The course is intended for clerks, typists, and others
who intend to use the UNIX system for preparing documentation.
The course will cover such topics as:
.VL 18
.LI Environment:
utilizing a time-sharing computer system;
accessing the system; using appropriate output terminals.
.LI Files:
how text is stored on the system; directories; manipulating files.
.LI "Text editing:"
how to enter text so that subsequent revisions are easier to make;
how to use the editing system to add, delete, and move lines of text;
how to make corrections.
.LI "Text processing:"
basic concepts; use of general purpose formatting packages.
.LI "Other facilities:"
additional capabilities useful to the typist such as the \fIspell\fR,
\fIdiff\fR, and \fIgrep\fR commands,
and a desk-calculator package.
.LE
.SG jrm
.NS 0
S. P. LeName
I. M. Here
U. R. There
R. Rhoade
.NE
```

Figure 2-3. Example of Input For a Simple Letter

Your Company

subject: Out-of-Hours Course Description -
Case 334455

date: May 31, 1983
from: D. W. Stevenson
PY 9876
1X-123 x5432

J. M. Jones:

Please use the following description for the out-of-hours course
Document Preparation on the UNIX* System:

Environment: utilizing a time-sharing computer system; accessing the system; using appropriate output terminals.

Files: how text is stored on the system; directories; manipulating files.

Text editing: how to enter text so that subsequent revisions are easier to make; how to use the editing system to add, delete, and move lines of text; how to make corrections.

Text processing: basic concepts; use of general-purpose formatting packages.

Other facilities: additional capabilities useful to the typist such as the spell, diff, and grep commands, and a desk-calculator package.

PY-9876-DWS-jrm D. W. Stevenson

Copy to
S. P. LeName
I. M. Here
U. R. There
R. Rhoade

---

* Trademark of Bell Laboratories

Figure 2-4. Example of *Nroff* Output for a Simple Letter

Your Company

subject: **Out-of-Hours Course Description - Case 334455**

date: **May 31, 1983**
from: **D. W. Stevenson**
PY 9876
1X-123 x5432

J. M. Jones:

Please use the following description for the out-of-hours course *Document Preparation on the UNIX* System*:

Environment:	utilizing a time-sharing computer system; accessing the system; using appropriate output terminals.
Files:	how text is stored on the system; directories; manipulating files.
Text editing:	how to enter text so that subsequent revisions are easier to make; how to use the editing system to add, delete, and move lines of text; how to make corrections.
Text processing:	basic concepts; use of general-purpose formatting packages.
Other facilities:	additional capabilities useful to the typist such as the *spell*, *diff*, and *grep* commands, and a desk-calculator package.

PY-9876-DWS-jrm **D. W. Stevenson**

Copy to
S. P. LeName
I. M. Here
U. R. There
R. Rhoade

* Trademark of Bell Laboratories

Figure 2-5. Example of *Troff* Output for a Simple Letter

6.11 End of Memorandum Macros

At the end of a memorandum document (but not of a released-paper document), signatures of authors and a list of notations can be requested. The following macros and their input are ignored if the released-paper style document is selected.

6.11.1 Signature Block

.FC [closing]
.SG [arg] [1]

The **.FC** macro prints "Yours very truly," as a formal closing, if no closing argument is used. It must be given before the .SG macro. A different closing may be specified as an argument to .FC.

The **.SG** macro prints the author's name(s) after the formal closing, if any. Each name begins at the center of the page. Three blank lines are left above each name for the actual signature.

- If no arguments are given, the line of reference data (location code, department number, author's initials, and typist's initials, all separated by hyphens) will not appear.
- A non-null first argument is treated as the typist's initials and is appended to the reference data.
- A null first argument prints reference data without the typist's initials or the preceding hyphen.
- If there are several authors and if the second argument is given, reference data is placed on the line of the first author.

Reference data contains only the location and department number of the first author. Thus, if there are authors from different departments and/or from different locations, the reference data should be supplied manually after the invocation (without arguments) of the .SG macro.

For example:

.SG
.rs
.sp -1v
PY/MH-9876/5432-JJJ/SPL-cen

6.11.2 "Copy to" and Other Notations

.NS [arg] [1]
zero or more lines of the notation
.NE

Many types of notations (such as a list of attachments or "Copy to" lists) may follow signature and reference data. Various notations are obtained through the **.NS** macro, which provides for proper spacing and for breaking notations across pages, if necessary.

The optional second argument, if present, causes the first argument to be used as the *entire* notation string. Codes for *arg* and the corresponding notations are:

arg	*NOTATIONS*
none	Copy to
" "	Copy to
0	Copy to
1	Copy (with att.) to
2	Copy (without att.) to
3	Att.
4	Atts.
5	Enc.
6	Encs.
7	Under Separate Cover
8	Letter to
9	Memorandum to
10	Copy (with atts.) to
11	Copy (without atts.) to
12	Abstract Only to
13	Complete Memorandum to
"*string*"	Copy (string) to
"*string*", with 2nd arg	string

If *arg* consists of more than one character, it is placed within parentheses between the words "Copy" and "to".

For example:

.NS " with att. 1 only"

will generate

Copy (with att. 1 only) to

as the notation.

More than one notation may be specified before the .NE macro because a .NS macro terminates the preceding notation, if any. For example:

.NS 4
Attachment 1-List of register names
Attachment 2-List of string and macro names
.NS 1
J. 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
J. J. Jones

Copy (without att.) to
S. P. LeName
G. H. Hurtz

If the second argument is used, then the first argument becomes the entire notation.

For example:

.NS" Table of Contents to" 1

would be formatted with

Table of Contents to

as the notation.

The .NS and .NE macros may also be used at the beginning following .AS 2 and .AE to place the notation list on the memorandum for file cover sheet {6.5}. If notations are given at the beginning without .AS 2, they will be saved and output at the end of the document.

6.11.3 Approval Signature Line

.A V approver's-name [1]

The **.AV** macro may be used after the last notation block to automatically generate a line with spaces for the approval signature and date. For example:

.A V" Jane Doe"

produces

APPROVED:

_____ _____

Jane Doe Date

The optional second argument, if present, prevents the "APPROVED:" mark from appearing above the approval line.

6.12 One-Page Letter

At times, the user may like more space on the page forcing the signature or items within notations to the bottom of the page so that the letter or memo is only one page in length. This can be accomplished by increasing the page length with the -rL*n* option, e.g., -rL90. This has the effect of making the formatter believe that the page is 90 lines long and therefore providing more space than usual to place the signature or the notations.

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

7. Displays

Displays are blocks of text that are to be kept together on a page and not split across pages. They are processed in an environment that is different from the body of the text (see the .ev request). The MM package provides two styles of displays: a *static* (.DS) style and a *floating* (.DF) style.

- In the *static* style, the display appears in the same relative position in the output text as it does in the input text. This may result in extra white space at the bottom of the page if the display is too long to fit in the remaining page space.

- In the *floating* style, the display "floats" through the input text to the top of the next page if there is not enough space on the current page. Thus input text that follows a floating display may precede it in the output text. A queue of floating displays is maintained so that their relative order of appearance in the text is not disturbed.

By default, a display is processed in no-fill mode with single spacing and is not indented from the existing margins. The user can specify indentation or centering as well as fill-mode processing.

> ***Note:*** Displays and footnotes {8} may never be nested in any combination. Although lists {5} and paragraphs {4.1} are permitted, no headings (.H or .HU) {4.2, 4.3} can occur within displays or footnotes.

7.1 Static Displays

.DS [format] [fill] [rindent]
one or more lines of text
.DE

A static display is started by the **.DS** macro and terminated by the **.DE** macro. With no arguments, .DS accepts lines of text exactly as typed (no-fill mode) and will not indent lines from the prevailing left margin indentation or from the right margin.

- The *format* argument is an integer or letter used to control the left margin indentation and centering with the following meanings:

format	*MEANING*
""	no indent
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
omitted	no indent

- The *fill* argument is an integer or letter and can have the following meanings:

fill	*MEANING*
""	no-fill mode
0 or N	no-fill mode
1 or F	fill mode
omitted	no-fill mode

- The *rindent* argument is the number of characters that the line length should be decreased, i.e., an indentation from the right margin. This number must be unscaled in the **nroff** formatter and is treated as *ens*. It may be scaled in the **troff** formatter or else defaults to *ems*.

The standard amount of static display indentation is taken from the *Si* register, a default value of five spaces. Thus, text of an indented display aligns with the first line of indented paragraphs, whose indent is contained in the *Pi* register {4.1}. Even though their initial values are the same (default values), these two registers are independent.

The display *format* argument value 3 (or CB) centers (horizontally) the entire display as a block (as opposed to .DS 2 and .DF 2 which center each line individually). All collected lines are left justified, and the display is centered based on width of the longest line. This format must be used in order for the **eqn/neqn** "mark" and "lineup" feature to work with centered equations {7.4}.

By default, a blank line (one-half a vertical space) is placed before and after *static* and *floating* displays. These blank lines before and after *static* displays can be inhibited by setting the register *Ds* to 0.

The following example shows usage of all three arguments for *static* displays. This block of text will be indented five spaces (ems in **troff**) from the left margin, filled, and indented five spaces (ems in **troff**) from the right margin (i.e., centered). The input text

```
.DS I F 5
"We the people of the United States,
in order to form a more perfect union,
establish justice, ensure domestic tranquillity,
provide for the common defense,
and secure the blessings of liberty to
ourselves and our posterity,
do ordain and establish this Constitution to the
United States of America."
.DE
```

produces

> *"We the people of the United States, in order to form a more perfect union, establish justice, ensure domestic tranquillity, provide for the common defense, and secure the blessings of liberty to ourselves and our posterity, do ordain and establish this Constitution to the United States of America."*

7.2 Floating Displays

```
.DF [format] [fill] [rindent]
one or more lines of text
.DE
```

A floating display is started by the **.DF** macro and terminated by the **.DE** macro. Arguments have the same meanings as static displays described above, except indent, no indent, and centering are calculated with respect to the initial left margin. This is because

prevailing indent may change between when the formatter first reads the floating display and when the display is printed. One blank line (one-half a vertical space) occurs before and after a floating display.

The user may exercise precise control over the output positioning of floating displays through the use of two number registers, *De* and *Df* (see below). When a floating display is encountered by the **nroff** or **troff** formatter, it is processed and placed onto a queue of displays waiting to be output. Displays are removed from the queue and printed in the order entered, which is the order they appeared in the input file. If a new floating display is encountered and the queue of displays is empty, the new display is a candidate for immediate output on the current page. Immediate output is governed by size of display and the setting of the *Df* register code. The *De* register code controls whether text will appear on the current page after a floating display has been produced.

As long as the display queue contains one or more displays, new displays will be automatically entered there, rather than being output. When a new page is started (or the top of the second column when in 2-column mode), the next display from the queue becomes a candidate for output if the *Df* register code has specified "top-of-page" output. When a display is output, it is also removed from the queue.

When the end of a section (using section-page numbering) or the end of a document is reached, all displays are automatically removed from the queue and output. This occurs before a .SG, .CS, or .TC macro is processed.

A display will fit on the current page if there is enough room to contain the entire display or if the display is longer than one page in length and less than half of the current page has been used. A wide (full-page width) display will not fit in the second column of a 2- column document.

The *De* and *Df* number register code settings and actions are as follows:

De REGISTER

CODE	*ACTION*
0	No special action occurs (also the default condition).
1	A page eject will always follow the output of each floating display, so only one floating display will appear on a page and no text will follow it.

Note: For any other code, the action performed is the same as for code 1.

Df REGISTER

CODE	*ACTION*
0	Floating displays will not be output until end of section (when section-page numbering) or end of document.
1	Output new floating display on current page if there is space; otherwise, hold it until end of section or document.
2	Output exactly one floating display from queue to the top of a new page or column (when in 2-column mode).
3	Output one floating display on current page if there is space; otherwise, output to the top of a new page or column.
4	Output as many displays as will fit (at least one) starting at the top of a new page or column. If the *De* register is set to 1, each display will be followed by a page eject causing a new top of page to be reached where at least one more display will be output.
5	Output a new floating display on the current page if there is room (default condition). Output as many displays (but at least one) as will fit on the page starting at the top of a new page or column. If the *De* register is set to 1, each display will be followed by a page eject causing a new top of page to be reached where at least one more display will be output.

Note: For any code greater than 5, the action performed is the same as for code 5.

The .WC macro {12.5} may also be used to control handling of displays in double-column mode and to control the break in text before floating displays.

7.3 Tables

.TS [H]
global options;
column descriptors.
title lines
[.TH [N]]
data within the table.
.TE

The **.TS** (table start) and **.TE** (table end) macros make possible the use of the **tbl** program. These macros are used to delimit text to be examined by **tbl** and to set proper spacing around the table. The display function and the **tbl** delimiting function are independent. In order to permit the user to keep together blocks that contain any mixture of tables, equations, filled text, unfilled text, and caption lines, the .TS/.TE block should be enclosed within a display (.DS/.DE). Floating tables may be enclosed inside floating displays (.DF/.DE).

Macros .TS and .TE permit processing of tables that extend over several pages. If a table heading is needed for each page of a multipage table, the "H" argument should be specified to the .TS macro as above. Following the options and format information, table title is typed on as many lines as required and is followed by the .TH macro. The .TH macro must occur when ".TS H" is used for a multipage table. This is not a feature of **tbl** but of the definitions provided by the MM macro package.

The **.TH** (table header) macro may take as an argument the letter **N**. This argument causes the table header to be printed only if it is the first table header on the page. This option is used when it is necessary to build long tables from smaller .TS H/.TE segments.

For example:

```
.TS H
global options;
column descriptors.
Title lines
.TH
data
.TE
.TS H
global options;
column descriptors.
Title lines
.TH N
data
.TE
```

will cause the table heading to appear at the top of the first table segment and no heading to appear at the top of the second segment when both appear on the same page. However, the heading will still appear at the top of each page that the table continues onto. This feature is used when a single table must be broken into segments because of table complexity (e.g., too many blocks of filled text). If each segment had its own .TS H/.TH sequence, it would have its own header. However, if each table segment after the first uses .TS H/.TH N, the table header will appear only at the beginning of the table and the top of each new page or column that the table continues onto.

For the **nroff** formatter, the -e option [-E for **mm** {2.1}] may be used for terminals, such as the 450, that are capable of finer printing resolution. This will cause better alignment of features such as the lines forming the corner of a box. The -e is not effective with *col.* (See *Preprocessor Reference* for more information on tables.)

7.4 Equations

.DS
.EQ [label]
equation(s)
.EN
.DE

Mathematical typesetting programs **eqn** and **neqn** expect to use the **.EQ** (equation start) and **.EN** (equation end) macros as delimiters in the same way that **tbl** uses .TS and .TE; however, .EQ and .EN must occur inside a .DS/.DE pair. There is an exception to this rule - if .EQ and .EN are used to specify only the delimiters for in-line equations or to specify **eqn/neqn** defines, the .DS and .DE macros must not be used; otherwise, extra blank lines will appear in the output.

The .EQ macro takes an argument that will be used as a label for the equation. By default, the label will appear at the right margin in the "vertical center" of the general equation. The *Eq* register may be set to 1 to change labeling to the left margin.

The equation will be centered for centered displays; otherwise, the equation will be adjusted to the opposite margin from the label.

7.5 Figure, Table, Equation, and Exhibit Titles

.FG [title] [override] [flag]
.TB [title] [override] [flag]
.EC [title] [override] [flag]
.EX [title] [override] [flag]

The **.FG** (figure title), **.TB** (table title), **.EC** (equation caption), and **.EX** (exhibit caption) macros are normally used inside .DS/.DE pairs to automatically number and title figures, tables, and equations.

These macros use registers *Fg*, *Tb*, *Ec*, and *Ex*, respectively (see paragraph 2.4 on -rN5 to reset counters in sections). For example:

.FG " This is a Figure Title"

yields

Figure 1. This is a Figure Title

The .TB macro replaces "Figure" with "TABLE", the .EC macro replaces "Figure" with "Equation", and the .EX macro replaces "Figure" with "Exhibit". The output title is centered if it can fit on a single line; otherwise, all lines but the first are indented to line up with the first character of the title. The format of the numbers may be changed using the **.af** request of the formatter. By setting the *Of* register to 1, the format of the caption may be changed from

Figure 1. Title

to

Figure 1 – Title

The *override* argument is used to modify normal numbering. If the *flag* argument is omitted or 0, *override* is used as a prefix to the number; if the *flag* argument is 1, *override* is used as a suffix; and if the *flag* argument is 2, *override* replaces the number. If -rN5 {2.4} is given, "section-figure" numbering is set automatically and user-specified *override* argument is ignored.

As a matter of formatting style, table headings are usually placed above the text of tables, while figure, equation, and exhibit titles are usually placed below corresponding figures and equations.

7.6 List of Figures, Tables, Equations, and Exhibits

A list of figures, tables, exhibits, and equations are printed following the table of contents if the number registers *Lf*, *Lt*, *Lx*, and *Le* (respectively) are set to 1. The *Lf*, *Lt*, and *Lx* registers are 1 by default; *Le* is 0 by default {18}.

Titles of these lists may be changed by redefining the following strings which are shown here with their default values:

```
.ds Lf LIST OF FIGURES
.ds Lt LIST OF TABLES
.ds Lx LIST OF EXHIBITS
.ds Le LIST OF EQUATIONS
```

8. Footnotes

There are two macros (.FS and .FE) that delimit text of footnotes, a string (F) that automatically numbers footnotes, and a macro (.FD) that specifies the style of footnote text. Footnotes are processed in an environment different from that of the body of text, refer to **.ev** request.

8.1 Automatic Numbering of Footnotes

Footnotes may be automatically numbered by typing the three characters "**F**" (i.e., invoking the string *F*) immediately after the text to be footnoted without any intervening spaces. This will place the next sequential footnote number (in a smaller point size) a half line above the text to be footnoted.

8.2 Delimiting Footnote Text

.FS [label]
one or more lines of footnote text
.FE

There are two macros that delimit the text of each footnote. The **.FS** (footnote start) macro marks the beginning of footnote text, and the **.FE** (footnote end) macro marks the end. The *label* on the .FS macro, if present, will be used to mark footnote text. Otherwise, the number retrieved from the string *F* will be used. Automatically numbered and user-labeled footnotes may be intermixed. If a footnote is labeled (.FS *label*), the text to be footnoted must be followed by *label*, rather than by "**F**". Text between .FS and .FE is processed in fill mode. Another .FS, a .DS, or a .DF are not permitted between .FS and .FE macros. If footnotes are required in the title, abstract, or table {7.3}, only labeled footnotes will appear properly. Everywhere else automatically numbered footnotes work correctly.

For example:

Automatically numbered footnote:

```
This is the line containing the word\*F
.FS
This is the text of the footnote.
.FE
to be footnoted.
```

Labeled footnote:

```
This is a labeled*
.FS *
The footnote is labeled with an asterisk.
.FE
footnote.
```

Text of the footnote (enclosed within the .FS/.FE pair) should immediately follow the word to be footnoted in the input text, so that "*F" or *label* occurs at the end of a line of input and the next line is the .FS macro call. It is also good practice to append an unpaddable space {3.3} to "*F" or *label* when they follow an end-of-sentence punctuation mark (i.e., period, question mark, exclamation point).

Figure 2-6 illustrates the various available footnote styles as well as numbered and labeled footnotes.

8.3 Format Style of Footnote Text

.FD [arg] [1]

Within footnote text, the user can control formatting style by specifying text hyphenation, right margin justification, and text indentation, as well as left or right justification of the label when text indenting is used. The **.FD** macro is invoked to select the appropriate style.

The first argument (**arg**) is a number from the left column of the following table. Formatting style for each number is indicated in the

remaining four columns. Further explanation of the first two of these columns is given in the definitions of the **.ad**, **.na**, **.hy**, and **.nh** (adjust, no adjust, hyphenation, and no hyphenation, respectively) requests.

arg	*HYPHENATION*	*ADJUST*	*TEXT INDENT*	*LABEL JUSTIFICATION*
0	.nh	.ad	yes	left
1	.hy	.ad	yes	left
2	.nh	.na	yes	left
3	.hy	.na	yes	left
4	.nh	.ad	no	left
5	.hy	.ad	no	left
6	.nh	.na	no	left
7	.hy	.na	no	left
8	.nh	.ad	yes	right
9	.hy	.ad	yes	right
10	.nh	.na	yes	right
11	.hy	.na	yes	right

If the first argument to .FD is greater than 11, the effect is as if **.FD 0** were specified. If the first argument is omitted or null, the effect is equivalent to **.FD 10** in the **nroff** formatter and to **.FD 0** in the **troff** formatter; these are also the respective initial default values.

If the second argument is specified, then when a first-level heading is encountered, automatically numbered footnotes begin again with 1. This is most useful with the "section-page" page numbering scheme. As an example, the input line

```
.FD "" 1
```

maintains the default formatting style and causes footnotes to be numbered afresh after each first-level heading in a document.

Hyphenation across pages is inhibited by MM except for long footnotes that continue to the following page. If hyphenation is permitted, it is possible for the last word on the last line on the current page footnote to be hyphenated. The user has control over this situation by specifying an even .FD argument.

Footnotes are separated from the body of the text by a short line rule. Those that continue to the next page are separated from the body of the text by a full-width rule. In the **troff** formatter, footnotes are set in type two points smaller than the point size used in the body of text.

8.4 Spacing Between Footnote Entries

Normally, one blank line (a 3-point vertical space) 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 (a 6-point vertical space) to occur between footnotes.

```
.FD 10
.P
This example illustrates several footnote styles
for both labeled and automatically numbered footnotes.
With the footnote style set to the \fBnroff\fR default style,
the first footnote is processed\*F
.FS
This is the first footnote text example.
This is the default style (.FD 10) for the \fBnroff\fR formatter.
The right margin is not justified,
hyphenation is not permitted,
text is indented, and the automatically generated label is
right justified in the text-indent space.
.FE
and followed by a second footnote.*****
.FS *****
This is the second footnote text example.
This is also the \fBnroff\fR formatter default style (.FD 10)
but with a long footnote label (*****) provided by the user.
.FE
.FD 1
Footnote style is changed by using the .FD macro to
specify hyphenation, right margin justification,
indentation, and left justification of the label.
This produces the third footnote,\*F
.FS
This is the third footnote example (.FD 1).
The right margin is justified, the footnote text is indented,
and the label is left justified in the text-indent space.
Although not necessarily illustrated by this example,
hyphenation is permitted.
.FE
and then the fourth footnote.\(dg
.FS \(dg
This is the fourth footnote example (.FD 1).
The style is the same as the third footnote.
.FE
.FD 6
Footnote style is set again via the .FD macro for no hyphenation,
no right margin justification,
no indentation, and with the label left justified.
This produces the fifth footnote.\*F
.FS
This is the fifth footnote example (.FD 6).
The right margin is not justified, hyphenation is not permitted,
footnote text is not indented,
and the label is placed at the beginning of the first line.
.FE
```

Figure 2-6. Example of Input for Various Footnote Styles

This example illustrates several footnote styles for both labeled and automatically numbered footnotes. With the footnote style set to the **nroff** default style, the first footnote is processed[1] and followed by a second footnote.***** Footnote style is changed by using the .FD macro to specify hyphenation, right margin justification, indentation, and left justification of the label. This produces the third footnote,[2] and then the fourth footnote.† Footnote style is set again via the .FD macro for no hyphenation, no right margin justification, no indentation, and with the label left justified. This produces the fifth footnote.[3]

Figure 2-7. Example of Output for Various Footnote Styles

---

1. This is the first footnote text example. This is the default style (.FD 10) for the **nroff** formatter. The right margin is not justified, hyphenation is not permitted, text is indented, and the automatically generated label is right justified in the text-indent space.

***** This is the second footnote text example. This is also the **nroff** formatter default style (.FD 10) but with a long footnote label (*****) provided by the user.

2. This is the third footnote example (.FD 1). The right margin is justified, the footnote text is indented, and the label is left justified in the text-indent space. Although not necessarily illustrated by this example, hyphenation is permitted.

† This is the fourth footnote example (.FD 1). The style is the same as the third footnote.

3. This is the fifth footnote example (.FD 6). The right margin is not justified, hyphenation is not permitted, footnote text is not indented, and the label is placed at the beginning of the first line.

9. Page Headers and Footers

Text printed at the top of each page is called *page header*. Text printed at the bottom of each page is called *page footer*. There can be up to three lines of text associated with the header - every page, even page only, and odd page only. Thus the page header may have up to two lines of text - the line that occurs at the top of every page and the line for the even- or odd-numbered page. The same is true for the page footer.

This part describes the default appearance of page headers and page footers and ways of changing them. The term *header* (not qualified by *even* or *odd*) is used to mean the page header line that occurs on every page, and similarly for the term *footer*.

9.1 Default Headers and Footers

By default, each page has a centered page number as the header. There is no default footer and no even/odd default headers or footers except as specified in paragraph 9.3.

In a memorandum or a released-paper style document, the page header on the first page is automatically suppressed provided a break does not occur before the .MT macro is called. Macros and text in the following categories do not cause a break and are permitted before the memorandum types (.MT) macro:

- Memorandum and released-paper style document macros (.TL, .AU, .AT, .TM, .AS, .AE, .OK, .ND, .AF, .NS, and .NE)
- Page headers and footers macros (.PH, .EH, .OH, .PF, .EF, and .OF)
- The **.nr** and **.ds** requests.

9.2 Header and Footer Macros

For header and footer macros (.PH .EH, .OH, .PF, .EF, and .OF) the argument [arg] is of the form:

" 'left-part'center-part'right-part' "

If it is inconvenient to use apostrophe (') as the delimiter because it occurs within one of the parts, it may be replaced uniformly by any other character. The **.fc** request redefines the delimiter. In formatted output, the parts are left justified, centered, and right justified, respectively.

9.2.1 Page Header

.PH [arg]

The **.PH** macro specifies the header that is to appear at the top of every page. The initial value is the default centered page number enclosed by hyphens. The page number contained in the *P* register is an Arabic number. The format of the number may be changed by the **.af** macro request.

If *"debug mode"* is set using the flag -rD1 on the command line {2.4}, additional information printed at the top left of each page is included in the default header. This consists of the Source Code Control System (SCCS) release and level of MM (thus identifying the current version {12.3}) followed by the current line number within the current input file.

9.2.2 Even-Page Header

.EH [arg]

The **.EH** macro supplies a line to be printed at the top of each even-numbered page immediately following the header. Initial value is a blank line.

9.2.3 Odd-Page Header

.OH [arg]

The **.OH** macro is the same as the .EH except that it applies to odd-numbered pages.

9.2.4 Page Footer

.PF [arg]

The **.PF** macro specifies the line that is to appear at the bottom of each page. Its initial value is a blank line. If the -rC*n* flag is specified on the command line {2.4}, the type of copy follows the footer on a separate line. In particular, if -rC3 or -rC4 (DRAFT) is specified, the footer is initialized to contain the date {6.8} instead of being a blank line.

9.2.5 Even-Page Footer

.EF [arg]

The **.EF** macro supplies a line to be printed at the bottom of each even-numbered page immediately preceding the footer. Initial value is a blank line.

9.2.6 Odd-Page Footer

.OF [arg]

The **.OF** macro supplies a line to be printed at the bottom of each odd-numbered page immediately preceding the footer. Initial value is a blank line.

9.2.7 First Page Footer

By default, the first page footer is a blank line. If, in the input text file, the user specifies .PF and/or .OF before the end of the first page of the document, these lines will appear at the bottom of the first page.

The header (whatever its contents) replaces the footer on the first page only if the -rN1 flag is specified on the command line {2.4}.

9.3 Default Header and Footer With Section-Page Numbering

Pages can be numbered sequentially within sections by "section-number page-number" {4.5}. To obtain this numbering style, -rN3 or -rN5 is specified on the command line. In this case, the default *footer* is a centered "section-page" number, e.g., 7-2; and the default page header is blank.

9.4 Strings and Registers in Header and Footer Macros

String and register names may be placed in arguments to header and footer macros. If the value of the string or register is to be computed when the respective header or footer is printed, invocation must be escaped by four backslashes. This is because string or register invocation will be processed three times:

1. As the argument to the header or footer macro
2. In a formatting request within the header or footer macro
3. In a **.tl** request during header or footer processing.

For example, page number register *P* must be escaped with four backslashes in order to specify a header in which the page number is to be printed at the right margin, e.g.:

```
.PH "''''Page \\\\nP'"
```

creates a right-justified header containing the word "Page" followed

by the page number. Similarly, to specify a footer with the "section-page" style, the user specifies (see paragraph 4.2.2.5 for meaning of *H1*):

```
.PF "''- \\\\n(H1-\\\\nP -' "
```

If the user arranges for the string *a]* to contain the current section heading which is to be printed at the bottom of each page, the .PF macro call would be:

```
.PF "''\\\\*(a]''"
```

If only one or two backslashes were used, the footer would print a constant value for *a],* namely, its value when .PF appeared in the input text.

9.5 Header and Footer Example

The following sequence specifies blank lines for header and footer lines, page numbers on the outside margin of each page (i.e., top left margin of even pages and top right margin of odd pages), and "Revision 3" on the top inside margin of each page (nothing is specified for the center):

```
.PH ""
.PF ""
.EH "'\\\\nP''Revision 3'"
.OH "'Revision 3''\\\\nP'"
```

9.6 Generalized Top-of-Page Processing

Note: This part is intended only for users accustomed to writing formatter macros.

During header processing, MM invokes two user-definable macros:

- The **.TP** (top of page) macro is invoked in the environment (refer to **.ev** request) of the header.
- The **.PX** is a page header user-exit macro that is invoked (without arguments) when the normal environment has been restored and with the "no-space" mode already in effect.

The effective initial definition of .TP (after the first page of a document) is

```
.de TP
.sp 3
.tl \\*(}t
.if e 'tl \\*(}e
.if o 'tl \\*(}o
.sp 2
..
```

The string *}t* contains the header, the string *}e* contains the even-page header, and the string *}o* contains the odd-page header as defined by the .PH, .EH, and .OH macros, respectively. To obtain more specialized page titles, the user may redefine the .TP macro to cause the desired header processing {12.6}. Formatting done within the .TP macro is processed in an environment different from that of

the body. For example, to obtain a page header that includes three centered lines of data, i.e., document number, issue date, and revision date, the user could define the .TP as follows:

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

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

9.7 Generalized Bottom-of-Page Processing

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

Lines of text that are specified between the **.BS** (bottom-block start) and **.BE** (bottom-block end) macros will be printed at the bottom of each page after the footnotes (if any) but before the page footer. This block of text is removed by specifying an empty block, i.e.:

```
.BS
.BE
```

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

9.8 Top and Bottom (Vertical) Margins

.VM [top] [bottom]

The **.VM** (vertical margin) macro allows the user to specify additional space at the top and bottom of the page. This space precedes the page header and follows the page footer. The .VM macro takes two unscaled arguments that are treated as v's. For example:

.VM 10 15

adds 10 blank lines to the default top of page margin and 15 blank lines to the default bottom of page margin. Both arguments must be positive (default spacing at the top of the page may be decreased by redefining .TP).

9.9 Proprietary Marking

.PM [code]

The **.PM** (proprietary marking) macro appends to the page footer a proprietary disclaimer. The *code* argument may be:

code	*DISCLAIMER*
none	turn off previous disclaimer, if any
P	PRIVATE
N	NOTICE
BP	BELL LABORATORIES PRIVATE
BPP (or BR)	BELL LABORATORIES PROPRIETARY - PRIVATE
BPN	BELL LABORATORIES - NOTICE
ILL	"RENDERED ILLEGIBLE" message
CI-II	Computer Inquiry II message

These disclaimers are in a form approved for use by the Bell System. The user may alternate disclaimers by use of the .BS/.BE macro pair.

Markings are underlined (italic in **troff**). The CI-II marking may be used with any other message by two separate .PM requests. For example:

```
.PM CI-II
.PM N
```

produces a CI-II *and* NOTICE mark.

9.10 Private Documents

.nr Pv value

The word "PRIVATE" may be printed, centered, and underlined on the second line of a document (preceding the page header). This is done by setting the *Pv* register *value*:

value	*MEANING*
0	do not print PRIVATE (default)
1	PRIVATE on first page only
1	PRIVATE on all pages

If *value* is 2, the user definable .TP macro may not be used because the .TP macro is used by MM to print "PRIVATE" on all pages except the first page of a memorandum on which .TP is not invoked.

10. Table of Contents and Cover Sheet

The table of contents and the cover sheet for a document are produced by invoking the .TC and .CS macros, respectively.

> ***Note:*** This section refers to cover sheets for technical memoranda and released papers only. The mechanism for producing a memorandum for file cover sheet was discussed earlier {6.5}.

These macros normally appear once at the end of the document, after the Signature Block {6.11.1} and Notations {6.11.2} macros, and may occur in either order.

The table of contents is produced at the end of the document because the entire document must be processed before the table of contents can be generated. Similarly, the cover sheet may not be desired by a user and is therefore produced at the end.

10.1 Table of Contents

.TC [slevel] [spacing] [tlevel] [tab] [h1] [h2] [h3] [h4] [h5]

The **.TC** macro generates a table of contents containing heading levels that were saved for the table of contents as determined by the value of the *Cl* register {4.4}. Arguments to .TC control spacing before each entry, placement of associated page number, and additional text on the first page of the table of contents before the word "CONTENTS".

Spacing before each entry is controlled by the first and second arguments (*slevel* and *spacing*). Headings whose level is less than or equal to *slevel* will have *spacing* blank lines (halves of a vertical space) before them. Both *slevel* and *spacing* default to 1. This means that first-level headings are preceded by one blank line (one-half a vertical space). The *slevel* argument does not control what levels of heading have been saved; saving of headings is the function of the *Cl* register.

The third and fourth arguments (*tlevel* and *tab*) control placement of associated page number for each heading. Page numbers can be justified at the right margin with either blanks or dots (called leaders) separating the heading text from the page number, or the page numbers can follow the heading text.

- For headings whose level is less than or equal to *tlevel* (default 2), page numbers are justified at the right margin. In this case, the value of *tab* determines the character used to separate heading text from page number. If *tab* is 0 (default value), dots (i.e., leaders) are used. If *tab* is greater than 0, spaces are used.

- For headings whose level is greater than *tlevel*, page numbers are separated from heading text by two spaces (i.e., page numbers are "ragged right", not right justified).

Additional arguments (*h1 ... h5*) 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 2
.in +5n
Approved: \l'2.5i'
.in
.sp
..
.TC
```

yields the following output when the file is formatted

Special Application
Message Transmission

Approved: _____

CONTENTS

.
.
.

If the .TX macro were defined as .TY, the word "CONTENTS" would be suppressed. Defining .TY as an empty macro will suppress "CONTENTS" with no replacement:

```
.de TY
..
```

By default, the first level headings will appear in the table of contents left justified. Subsequent levels will be aligned with the text of headings at the preceding level. These indentations may be

changed by defining the *Ci* string which takes a maximum of seven arguments corresponding to the heading levels. It must be given at least as many arguments as are set by the *Cl* register. Arguments must be scaled.

For example, with "*Cl* = 5":

```
.ds Ci .25i .5i .75i 1i 1i  \" troff
```

or

```
.ds Ci 0 2n 4n 6n 8n  \" nroff
```

Two other registers are available to modify the format of the table of contents - *Oc* and *Cp*.

- By default, table of contents pages will have lowercase Roman numeral page numbering. If the *Oc* register is set to 1, the .TC macro will not print any page number but will instead reset the *P* register to 1. It is the user's responsibility to give an appropriate page footer to specify the placement of the page number. Ordinarily, the same .PF macro (page footer) used in the body of the document will be adequate.

- The list of figures, tables, etc. pages will be produced separately unless *Cp* is set to 1 which causes these lists to appear on the same page as the table of contents.

10.2 Cover Sheet

```
.CS [pages] [other] [total] [figs] [tbls] [refs]
```

The **.CS** macro generates a cover sheet in either the released paper or technical memorandum style (see paragraph 6.5 for details of the memorandum for file cover sheet). All other information for the cover sheet is obtained from data given before the .MT macro call {6.1}. If the technical memorandum style is used, the .CS macro generates the "Cover Sheet for Technical Memorandum". The data that appear in the lower left corner of the technical memorandum

cover sheet (counts of: pages of text, other pages, total pages, figures, tables, and references) are generated automatically (0 is used for "other pages"). These values may be changed by supplying the corresponding arguments to the .CS macro. If the released-paper style is used, all arguments to .CS are ignored.

11. References

There are two macros (.RS and .RF) that delimit the text of references, a string that automatically numbers the subsequent references, and an optional macro (.RP) that produces reference pages within the document.

11.1 Automatic Numbering of References

Automatically numbered references may be obtained by typing ***(Rf** (invoking the string *Rf*) immediately after the text to be referenced. This places the next sequential reference number (in a smaller point size) enclosed in brackets one-half line above the text to be referenced. Reference count is kept in the *Rf* number register. The number register actually used to print the reference number for each reference call (*(Rf) in the text is *:R*. The *:R* register may have its format or value changed to effect the reference marks, without affecting the total count of references.

11.2 Delimiting Reference Text

```
.RS [string-name]
.RF
```

The **.RS** and **.RF** macros are used to delimit text of each reference as shown below:

```
A line of text to be referenced.\*(Rf
.RS
reference text
.RF
```

11.3 Subsequent References

The **.RS** macro takes one argument, a *string-name*. For example:

```
.RS aA
reference text
.RF
```

The string *aA* is assigned the current reference number. This string may be used later in the document as the string call, *(*aA*, to reference text which must be labeled with a prior reference number. The reference is output enclosed in brackets one-half line above the text to be referenced. No .RS/.RF pair is needed for subsequent references.

11.4 Reference Page

.RP [arg1] [arg2]

A reference page, entitled by default "REFERENCES", will be generated automatically at the end of the document (before table of contents and cover sheet) and will be listed in the table of contents. This page contains the reference items (i.e., reference text enclosed within .RS/.RF pairs). Reference items will be separated by a space (one-half a vertical space) unless the *Ls* register is set to 0 to suppress this spacing. The user may change the reference page title by defining the *Rp* string:

.ds Rp " New Title"

The **.RP** (reference page) macro may be used to produce reference pages anywhere else within a document (i.e., after each major section). It is not needed to produce a separate reference page with default spacings at the end of the document.

Two .RP macro arguments allow the user to control resetting of reference numbering and page skipping.

arg1	*MEANING*
0	reset reference counter (default)
1	do not reset reference counter

arg2	*MEANING*
0	put on separate page (default)
1	do not cause a following .SK
2	do not cause a preceding .SK
3	no .SK before or after

If no .SK macro is issued by the .RP macro, a single blank line will separate the references from the following/preceding text. The user may wish to adjust spacing. For example, to produce references at the end of each major section:

```
.sp 3
.RP 1 2
.H 1 " Next Section"
```

12. Miscellaneous Features

12.1 Bold, Italic, and Roman Fonts

```
.B [bold-arg] [previous-font-arg] ...
.I [italic-arg] [previous-font-arg] ...
.R
```

When called without arguments, the **.B** macro changes the font to bold and the **.I** macro changes to underlining (italic). This condition continues until the occurrence of the **.R** macro which causes the Roman font to be restored. Thus:

```
.I
here is some text.
.R
```

yields underlined text via the **nroff** and italic text via the **troff** formatter.

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 italic " <sp>text<sp>"  right -justified
<sp> indicates a space
```

produces

italic text *right*-justified

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 .BI .IR .RI .RB .BR

Each macro takes a maximum of six arguments and alternates arguments between specified fonts.

When using a terminal that cannot underline, the following can be inserted at the beginning of the document to eliminate all underlining:

```
.rm ul
.rm cu
```

Note: Font changes in headings are handled separately {4.2.2.4.1}.

12.2 Justification of Right Margin

.SA [arg]

The **.SA** macro is used to set right-margin justification for the main body of text. Two justification flags are used - *current* and *default.* Initially, both flags are set for no justification in the **nroff** formatter and for justification in the **troff** formatter. The argument causes the following action:

arg	*MEANING*
0	Sets both flags to no justification. It acts like the **.na** request.
1	Sets both flags to cause both right and left justification, the same as the **.ad** request.
omitted	Causes the current flag to be copied from the default flag, thus performing either a **.na** or **.ad** depending on the default condition.

In general, the no adjust request (**.na**) can be used to ensure that justification is turned off, but .SA should be used to restore justification, rather than the **.ad** request. In this way, justification or no justification for the remainder of the text is specified by inserting ".SA 0" or ".SA 1" once at the beginning of the document.

12.3 SCCS Release Identification

The *RE* string contains the SCCS release and the MM text formatting macro package current version level. For example:

```
This is version \*(RE of the macros.
```

produces

```
This is version 10.129 of the macros.
```

This information is useful in analyzing suspected bugs in MM. The easiest way to have the release identification number appear in the output is to specify -rD1 {2.4} on the command line. This causes the *RE* string to be output as part of the page header {9.2.1}.

12.4 Two-Column Output

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

The MM text formatting macro package can format two-columns on a page. The **.2C** macro begins 2-column processing which continues until a **.1C** macro (1-column processing) is encountered. In 2-column processing, each physical page is thought of as containing 2-columnar "pages" of equal (but smaller) "page" width. Page headers and footers are not affected by 2-column processing. The .2C macro does not balance 2-column output.

12.5 Footnotes and Displays for Two-Column Output

It is possible to have full-page width footnotes and displays when in 2-column mode, although default action is for footnotes and displays to be narrow in 2-column mode and wide in 1-column mode. Footnote and display width is controlled by the **.WC** (width control) macro, which takes the following arguments:

arg	*MEANING*
N	Default mode (-WF, -FF, -WD, FB).
WF	Wide footnotes (even in 2-column mode).
-WF	DEFAULT: Turn off WF. Footnotes follow column mode; wide in 1-column mode (1C), narrow in 2-column mode (2C), unless FF is set.
FF	First footnote. All footnotes have same width as first footnote encountered for that page.
-FF	DEFAULT: Turn off FF. Footnote style follows settings of WF or -WF.
WD	Wide displays (even in 2-column mode).
-WD	DEFAULT: Displays follow the column mode in effect when display is encountered.
FB	DEFAULT: Floating displays cause a break when output on the current page.
-FB	Floating displays on current page do not cause a break.

Note: The **.WC WD FF** command will cause all displays to be wide and all footnotes on a page to be the same width while **.WC N** will reinstate default actions. If conflicting settings are given to **.WC**, the last one is used. A **.WC WF −WF** command has the effect of a **.WC −WF**.

12.6 Column Headings for Two-Column Output

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

In 2-column processing output, it is sometimes necessary to have headers over each column, as well as headers over the entire page. This is accomplished by redefining the .TP macro {9.6} to provide header lines both for the entire page and for each of the columns. For example:

```
.de TP
.sp 2
.tl 'Page \\nP'OVERALL"
.tl "TITLE"
.sp
.nf
.ta 16C 31R 34 50C 65R
left⟨T⟩center⟨T⟩right⟨T⟩left⟨T⟩center⟨T⟩right
⟨T⟩first column⟨T⟩⟨T⟩⟨T⟩second column
.fi
.sp 2
..
```

where ⟨T⟩ stands for the tab character.

The above example will produce two lines of page header text plus two lines of headers over each column. Tab stops are for a 65-en overall line length.

12.7 Vertical Spacing

.SP [lines]

There exists several ways of obtaining vertical spacing, all with different effects. The **.sp** request spaces the number of lines specified unless the no space (**.ns**) mode is on, then the **.sp** request is ignored. The no space mode is set at the end of a page header to eliminate spacing by a **.sp** or **.bp** request that happens to occur at

the top of a page. This mode can be turned off by the **.rs** (restore spacing) request.

The **.SP** macro is used to avoid the accumulation of vertical space by successive macro calls. Several .SP calls in a row will not produce the sum of the arguments but only the maximum argument. For example, the following produces only three blank lines:

```
.SP 2
.SP 3
.SP
```

Many MM macros utilize .SP for spacing. For example, ".LE 1" {5.1.3} immediately followed by ".P" {4.1} produces only a single blank line (one-half a vertical space) between the end of the list and the following paragraph. An omitted argument defaults to one blank line (one vertical space). 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** request.

12.8 Skipping Pages

```
.SK [pages]
```

The **.SK** macro skips pages but retains the usual header and footer processing. If the *pages* argument is omitted, null, or 0, .SK skips to the top of the next page unless it is currently at the top of a page (then it does nothing). A ".SK *n*" command skips *n* pages. A ".SK" positions text that follows it at the top of a page, while ".SK 1" leaves one page blank except for the header and footer.

12.9 Forcing an Odd Page

```
.OP
```

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

- If currently at the top of an odd-numbered page, text output begins on that page (no motion takes place).

- If currently on an even page, text resumes printing at the top of the next page.

- If currently on an odd page (but not at the top of the page), one blank page is produced, and printing resumes on the next odd-numbered page after that.

12.10 Setting Point Size and Vertical Spacing

.S [point size] [vertical spacing]

The prevailing point size and vertical spacing may be changed by invoking the **.S** macro. In the **troff** formatter, the default point size (obtained from the MM register *S* {2.4}) is 10 points, and the vertical spacing is 12 points (six lines per inch). The mnemonics D (default value), C (current value), and P (previous value) may be used for both arguments.

- If an argument is *negative*, current value is decremented by the specified amount.

- If an argument is *positive*, current value is incremented by the specified amount.

- If an argument is *unsigned*, it is used as the new value.

- If there are no arguments, the .S macro defaults to P.

- If the first argument is specified but the second is not, then (default) D is used for the vertical spacing.

Default value for vertical spacing is always two points greater than the current point size. Footnotes {8} are two points smaller than the

body with an additional 3-point space between footnotes. A null ("") value for either argument defaults to C (current value). Thus, if *n* is a numeric value:

```
.S          =   .S P P
.S "" n     =   .S C n
.S n ""     =   .S n C
.S n        =   .S n D
.S ""       =   .S C D
.S "" ""    =   .S C C
.S n n      =   .S n n
```

If the first argument is greater than 99, the default point size (10 points) is restored. If the second argument is greater than 99, the default vertical spacing (current point size plus two points) is used. For example:

```
.S 100      =   .S 10 12
.S 14 111   =   .S 14 16
```

12.11 Reducing Point Size of a String

.SM string1 [string2] [string3]

The **.SM** macro allows the user to reduce by one point the size of a string. If the third argument (*string3*) is omitted, the first argument (*string1*) is made smaller and is concatenated with the second argument (*string2*) if specified. If all three arguments are present (even if any are null), the second argument is made smaller and all three arguments are concatenated. For example:

INPUT	*OUTPUT*
.SM X	X
.SM X Y	XY
.SM Y X Y	YXY
.SM YXYX	YXYX
.SM YXYX)	YXYX)
.SM (YXYX)	(YXYX)
.SM Y XYX ""	YXYX

12.12 Producing Accents

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

	INPUT	*OUTPUT*
Grave accent	c*`	c̀
Acute accent	e*'	é
Circumflex	o*^	ô
Tilde	n*~	ñ
Cedilla	c*,	ç
Lower-case umlaut	u*:	ü
Upper-case umlaut	U*;	Ü

The string definitions that generated these letters are:

```
.ds ` \\k:\h@-\\n(.wu*8u/10u@\h@\\n(.fu/2u*2u+1u-\\n(.fu*.2m@\
\(ga\h@¦\\n:u@
.ds ' \\k:\h@-\\n(.wu*8u/10u@\h@\\n(.fu/2u*2u+1u-\\n(.fu*.2m+.07m@\
\(aa\h@¦\\n:u@
.ds ^ \\k:\h@-\\n(.wu*8u/10u@\h@\\n(.fu/2u*2u+1u-\\n(.fu*.15m-.07m@\
\h@\\n(.fu-1u/2u*.02m@^\h@¦\\n:u@
.ds ~ \\k:\h@-\\n(.wu*8u/10u@\h@\\n(.fu/2u*2u+1u-\\n(.fu*.2m-.07m@\
\h@\\n(.fu-1u/2u*.05m@~\h@¦\\n:u@
.ds , \\k:\h@-\\n(.wu*85u/100u@\v@.07m@,\v@-.07m@\h@¦\\n:u@
.ds : \\k:\h@-\\n(.wu*85u/100u@\h@\\n(.fu/2u*2u+1u-\\n(.fu*3u*.06m@\
\h@3u-\\n(.fu/2u*.05m-.1m@\
\v@-.6m@\z.\h@\\n(.fu-1u/2u*.05m+.2m@.\v@.6m@\h@¦\\n:u@
```

```
.ds ; \\k:\h@-\\n(.wu*75u/100u@\h@\\n(.fu/2u*2u+1u-\\n(.fu*3u*.09m@\
\h@3u-\\n(.fu/2u*.06m-.15m@\h@\\n(.fu-1u/2u*.04m@\
\v@-.85m@\z.\h@.3m@.\v@.85m@\h@¦\\n:u@
.if t .ds < \h@.05m@\s+4\v@.333m@\'\v@-.333m@\s-4\h@.05m@
.if n .ds < '
.if t .ds > \h@.05m@\s+4\v@.333m@\`\v@-.333m@\s-4\h@.05m@
.if n .ds > `
```

12.13 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: J. Jones (user types name)
16 Elm Rd.,
Piscataway

The diverted macro .aA will contain

J. Jones
16 Elm Rd.,
Piscataway

The string *bB* (*(*bB*) contains "J. Jones".

A newline character followed by an EOF (user specifiable CONTROL d) also allows the user to resume normal output.

13. Errors and Debugging

13.1 Error Terminations

When a macro detects an error, the following actions occur:

- A break occurs.
- The formatter output buffer (which may contain some text) is printed to avoid confusion regarding location of the error.
- A short message is printed giving the name of the macro that detected the error, type of error, and approximate line number in the current input file of the last processed input line. Error messages are explained in the last section of this chapter.
- Processing terminates unless register D {2.4} has a positive value. In the latter case, processing continues even though the output is guaranteed to be deranged from that point on.

The error message is printed by outputting the message directly to the user terminal. If an output filter, such as **300**, **450**, or **hp** is being used to post-process the **nroff** formatter output, the message may be garbled by being intermixed with text held in that filter's output buffer.

> ***Note:*** If any of **ocw**, **eqn/neqn**, and **tbl** programs are being used and if the **-o***list* option of the formatter causes the last page of the document not to be printed, a harmless "broken pipe" message may result.

13.2 Disappearance of Output

Disappearance of output usually occurs because of an unclosed diversion (e.g., a missing .DE or .FE macro). Fortunately, macros that use diversions are careful about it, and these macros check to make sure that illegal nestings do not occur. If any error message is issued concerning a missing .DE or .FE, the appropriate action is to search backwards from the termination point looking for the corresponding associated .DF, .DS, or .FS (since these macros are used in pairs).

The following command:

```
grep -n '^\.[EDFRT][EFNQS]' files ...
```

prints all the .DF, .DS, .DE, .EQ, .EN, .FS, .FE, .RS, .RF, .TS, and .TE macros found in *files* ..., each preceded by its file name and the line number in that file. This listing can be used to check for illegal nesting and/or omission of these macros.

14. Extending and Modifying MM Macros

14.1 Naming Conventions

In this part, the following conventions are used to describe names:

- n: Digit
- a: Lowercase letter
- A: Uppercase letter
- x: Any alphanumeric character (n, a, or A, i.e., letter or digit)
- s: Any nonalphanumeric character (special character)

All other characters are literals (characters that stand for themselves).

Request, macro, and string names are kept by the formatters in a single internal table; therefore, there must be no duplication among such names. Number register names are kept in a separate table.

14.1.1 Names Used by Formatters

requests:	aa (most common) an (only one, currently: c2)
registers:	aa (normal) .x (normal) .s (only one, currently: .$) a. (only one, currently: c.) % (page number)

14.1.2 Names Used by MM

macros and strings:	A, AA, Aa (accessible to users; e.g., macros P and HU, strings F, BU, and Lt) nA (accessible to users; only two, currently: 1C and 2C)

	aA (accessible to users; only one, currently: nP)
	s (accessible to users; only the seven accents, currently {12.10})
	)x, }x,]x, >x, ?x (internal)
registers:	An, Aa (accessible to users; e.g., H1, Fg)
	A (accessible to users; meant to be set on the command line; e.g., C)
	:x, ;x, #x, ?x, !x (internal)

14.1.3 Names Used by ocw, eqn/neqn, and tbl

The *ocw* program is the constant-width font preprocessor for the **otroff** formatter. It uses the following five macro names:

.CD .CN .CP .CW .PC

This preprocessor also uses the number register names *cE* and *cW*.

> ***Note:*** The *ocw* preprocessor cannot be used with device-independent **troff**.

Mathematical equation preprocessors, **eqn** and **neqn**, use registers and string names of the form *nn*. The table preprocessor, **tbl**, uses T&, T#, and TW, and names of the form:

a- a+ a! nn na ^a #a #s

14.1.4 Names Defined by User

Names that consist either of a single lowercase letter or a lowercase letter followed by a character other than a lowercase letter (names

.c2 and .nP are already used) should be used to avoid duplication with already used names. The following is a possible naming convention:

macros:	aA (e.g., bG, kW)
strings:	as (e.g., c), f], p})
registers:	a (e.g., f, t)

14.2 Sample Extensions

14.2.1 Appendix Headings

The following is a way of generating and numbering appendix headings:

```
.nr Hu 1
.nr a 0
.de aH
.nr a +1
.nr P 0
.PH " '''Appendix \\na-\\\\\\\\nP' "
.SK
.HU " \\$1"
..
```

After the above initialization and definition, each call of the form

```
.aH " title"
```

begins a new page (with the page header changed to "Appendix *a-n* ") and generates an unnumbered heading of *title*, which, if desired, can be saved for the table of contents. To center appendix titles the *Hc* register must be set to 1 {4.2.2.3}.

14.2.2 Hanging Indent With Tabs.

The following example illustrates the use of the hanging indent feature of variable-item lists {5.1.1.6}. In this example, a user-defined macro is defined to accept four arguments that make up the *mark*. In the output, each argument is to be separated from the

previous one by a tab; tab settings are defined later. Since the first argument may begin with a period or apostrophe, the "\&" is used so that the formatter will not interpret such a line as a formatter request or macro call.

> ***Note:*** The 2-character sequence "\&" is understood by formatters to be a "zero-width" space. It causes no output characters to appear, but it removes the special meaning of a leading period or apostrophe.

The "\t" is translated by the formatter into a tab. The "\c" is used to concatenate the input text that follows the macro call to the line built by the macro. The user-defined macro and an example of its use are:

```
.de aX
.LI
\&\\$1\t\\$2\t\\$3\t\\$4\t\c
..
  .
  .
  .
.ta 8 14 20 24
.VL 24
.aX .nh off \- no
No hyphenation.
Automatic hyphenation is turned off.
Words containing hyphens
(e.g., mother-in-law) may still be split across lines.
.aX .hy on \- no
Hyphenate.
Automatic hyphenation is turned on.
.aX .hc\<sp>c none none no
Hyphenation indicator character is set to "c" or removed.
During text processing, the indicator is suppressed
and will not appear in the output.
Prepending the indicator to a word has the effect
of preventing hyphenation of that word.
.LE
```

where <sp> stands for a space.

The resulting output is:

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

15. Summary

The following are qualities of MM that have been emphasized in its design in approximate order of importance:

- *Robustness in the face of error* — A user need not be an **nroff/troff** expert to use MM macros. When the input is incorrect, either the macros attempt to make a reasonable interpretation of the error or an error message describing the error is produced. An effort has been made to minimize the possibility that a user would get cryptic system messages or strange output as a result of simple errors.

- *Ease of use for simple documents* — It is not necessary to write complex sequences of commands to produce documents. Reasonable macro argument default values are provided where possible.

- *Parameterization* — There are many different preferences in the area of document styling. Many parameters are provided so that users can adapt input text files to produce output documents to their respective needs over a wide range of styles.

- *Extension by moderately expert users* — A strong effort has been made to use mnemonic naming conventions and consistent techniques in construction of macros. Naming conventions are given so that a user can add new macros or redefine existing ones if necessary.

- *Device independence* — A common use of MM is to produce documents on hard copy via teletypewriter terminals using the **nroff** formatter. Macros can be used conveniently with both 10- and 12-pitch terminals. In addition, output can be displayed on an appropriate CRT terminal. Macros have been constructed to allow compatibility with the **troff** formatter so that output can be produced on both a phototypesetter and a teletypewriter/CRT terminal.

- *Minimization of input* — The design of macros attempts to minimize repetitive typing. For example, if a user wants to have a blank line after all first- or second-level headings, the user need only set a specific parameter once at the beginning of a document rather than type a blank line after each such heading.

- *Decoupling of input format from output style* — There is but one way to prepare the input text although the user may obtain a number of output styles by setting a few global flags. For example, the .H macro is used for all numbered headings, yet the actual output style of these headings may be made to vary from document to document or within a single document.

16. MM Macro Name Summary

The following listing shows all the MM macros and their usage. Each item in the list gives a definition of the macro followed by its normal format and arguments. The numbers enclosed in braces {} indicate the paragraph or section in the first part of this chapter where a complete explanation of each macro may be found.

> ***Note:*** Macros marked with an asterisk are not, in general, called (invoked) directly by the user. They are "user exits" defined by the user and called by the MM macros from inside header, footer, or other macros.

1C — 1-column processing {12.4}
.1C

2C — 2-column processing {12.4}
.2C

AE — Abstract end {6.5}
.AE

AF — Alternate format of "Subject/Date/From" block {6.9}
.AF [company-name]

AL — Automatically incremented list start {5.1.1.1}
.AL [type] [text-indent] [1]

AS — Abstract start {6.5}
.AS [arg] [indent]

AT — Author's title {6.3}
.AT [title] ...

AU — Author information {6.3}
.AU name [initials] [loc] [dept] [ext] [room] [arg] [arg] [arg]

AV — Approval signature {6.11.3}
.AV [name] [1]

B — Bold {12.1}
.B [bold-arg] [previous-font-arg] [bold] [prev] [bold] [prev]

BE — Bottom block end {9.7}
.BE

BI — Bold/Italic {12.1}
.BI [bold-arg] [italic-arg] [bold] [italic] [bold] [italic]

BL — Bullet list start {5.1.1.2}
.BL [text-indent] [1]

BR — Bold/Roman {12.1}
.BR [bold-arg] [Roman-arg] [bold] [Roman] [bold] [Roman]

BS — Bottom block start {9.7}
.BS

CS — Cover sheet {10.2}
.CS [pages] [other] [total] [figs] [tbls] [refs]

DE — Display end {7.1}
.DE

DF — Display floating start {7.2}
.DF [format] [fill] [right-indent]

DL — Dash list start {5.1.1.3}
.DL [text-indent] [1]

DS — Display static start {7.1}
.DS [format] [fill] [right-indent]

EC — Equation caption {7.5}
.EC [title] [override] [flag]

EF — Even-page footer {9.2.5}
.EF [arg]

EH Even-page header {9.2.2}
.EH [arg]

EN End equation display {7.4}
.EN

EQ Equation display start {7.4}
.EQ [label]

EX Exhibit caption {7.5}
.EX [title] [override] [flag]

FC Formal closing {6.11.1}
.FC [closing]

FD Footnote default format {8.3}
.FD [arg] [1]

FE Footnote end {8.2}
.FE

FG Figure title {7.5}
.FG [title] [override] [flag]

FS Footnote start {8.2}
.FS [label]

H Heading—numbered {4.2}
.H level [heading-text] [heading-suffix]

HC Hyphenation character {3.4}
.HC [hyphenation-indicator]

HM Heading mark style (Arabic or Roman numerals, or letters) {4.2.2.5}
.HM [arg1] ... [arg7]

HU Heading—unnumbered {4.3}
.HU heading-text

HX* Heading user exit X (before printing heading) {4.6}
.HX dlevel rlevel heading-text

HY* Heading user exit Y (before printing heading) {4.6}
.HY dlevel rlevel heading-text

HZ* Heading user exit Z (after printing heading) {4.6}
.HZ dlevel rlevel heading-text

I Italic (underline in the **nroff** formatter) {12.1}
.I [italic-arg] [previous-font-arg] [italic] [prev] [italic] [prev]

IB Italic/Bold {12.1}
.IB [italic-arg] [bold-arg] [italic] [bold] [italic] [bold]

IR Italic/Roman {12.1}
.IR [italic-arg] [Roman-arg] [italic] [Roman] [italic] [Roman]

LB List begin {5.2}
.LB text-indent mark-indent pad type [mark] [LI-space] [LB-space]

LC List-status clear {5.3}
.LC [list-level]

LE List end {5.1.3}
.LE [1]

LI List item {5.1.2}
.LI [mark] [1]

ML Marked list start {5.1.1.4}
.ML mark [text-indent] [1]

MT Memorandum type {6.7}
.MT [type] [addressee] *or* .MT [4] [1]

ND New date {6.8}
.ND new-date

NE Notation end {6.11.2}
.NE

NS Notation start {6.11.2}
.NS [arg] [1] nP" Double-line indented paragraphs {4.1.2}
.nP

OF Odd-page footer {9.2.6}
.OF [arg]

OH Odd-page header {9.2.3}
.OH [arg]

OK Other keywords for the Technical Memorandum cover sheet {6.6}
.OK [keyword] ...

OP Odd page {12.9}
.OP

P Paragraph {4.1}
.P [type]

PF Page footer {9.2.4}
.PF [arg]

PH Page header {9.2.1}
.PH [arg]

PM Proprietary marking {9.9}
.PM [code]

PX* Page-header user exit {9.6}
.PX

R Return to regular (Roman) font {12.1}
.R

RB Roman/Bold {12.1}
.RB [Roman-arg] [bold-arg] [Roman] [bold] [Roman] [bold]

RD Read insertion from terminal {12.13}
.RD [prompt] [diversion] [string]

RF Reference end {11.2}
.RF

RI Roman/Italic {12.1}
.RI [Roman-arg] [italic-arg] [Roman] [italic] [Roman] [italic]

RL Reference list start {5.1.1.5}
.RL [text-indent] [1]

RP Produce reference page {11.4}
.RP [arg] [arg]

RS Reference start {11.2}
.RS [string-name]

S Set **troff** formatter point size and vertical spacing {12.10}
.S [size] [spacing]

SA Set adjustment (right-margin justification) default {12.2}
.SA [arg]

SG Signature line {6.11.1}
.SG [arg] [1]

SK Skip pages {12.8}
.SK [pages]

SM Make a string smaller {12.10}
.SM string1 [string2] [string3]

SP Space vertically {12.7}
.SP [lines]

TB Table title {7.5}
.TB [title] [override] [flag]

TC Table of contents {10.1}
.TC [slevel] [spacing] [tlevel] [tab] [h1] [h2] [h3] [h4] [h5]

TE Table end {7.3}
.TE

TH Table header {7.3}
.TH [N]

TL Title of memorandum {6.2}
.TL [charging-case] [filing-case]

TM Technical Memorandum number(s) {6.4}
.TM [number] ...

TP* Top-of-page macro {9.6}
.TP

TS Table start {7.3}
.TS [H]

TX* Table of contents user exit {10.1}
.TX

TY* Table of contents user exit (suppresses "CONTENTS") {10.1}
.TY

VL Variable-item list start {5.1.1.6}
.VL text-indent [mark-indent] [1]

VM Vertical margins {9.8}
.VM [top] [bottom]

WC Footnote and display width control {12.5}
.WC [format]

17. MM String Name Summary

The following list shows the predefined string names used by the MM macro package. The numbers in braces {} indicate the paragraph or section number in the first part of this chapter where more information about the string can be found.

BU Bullet {3.7}
NROFF: ⊕
TROFF: •

Ci Table of contents indent list {10.1}
Up to seven scaled arguments for heading levels

DT Date {6.8}
Current date, unless overridden
Month, day, year (e.g., May 31, 1979)

EM Em dash string {3.8}
Produces an em dash in the **troff** formatter and a double hyphen in **nroff**

F Footnote number generator {8.1}
NROFF: \u\\n+(:p\d
TROFF: \v'-.4m'\s-3\\n+(:p\s0\v'.4m'

HF Heading font list {4.2.2.4.1}
Up to seven codes for heading levels 1 through 7
2 2 2 2 2 2 2 (all levels underlined by **nroff** and italicized by **troff**)

HP Heading point size list {4.2.2.4.3}
Up to seven codes for heading levels 1 through 7

Le Title for LIST OF EQUATIONS {7.6}

Lf Title for LIST OF FIGURES {7.6}

Lt Title for LIST OF TABLES {7.6}

Lx Title for LIST OF EXHIBITS {7.6}

RE SCCS Release and Level of MM {12.3}
Release.Level (e.g., 15.129)

Rf Reference number generator {11.1}

Rp Title for References {11.4}

Tm Trademark string {3.9}
Places the letters "TM" one-half line above the text that it follows

Seven accent strings are also available. {12.12}

Note 1: If the released-paper style is used, then (in addition to the above strings) certain BTL location codes are defined as strings. These location strings are needed only until the .MT macro is called {6.7}. Currently, the following codes are recognized:

AK, AL, ALF, CB, CH, CP, DR, FJ, HL, HO, HOH, HP, IH, IN, INH, IW, MH, MV, PY, RD, RR, WB, WH, and WV.

Note 2: Paragraph 1.6 has notes on setting and referencing strings.

18. MM Number Register Summary

The list that follows contains a description of all the predefined number registers used by MM. Numbers enclosed in braces {} indicate the paragraph or section number in the first part of this chapter where more information about the register can be found. After each description is the normal value of the register followed by the range of allowable values, enclosed in brackets []. The lower and upper limit of values are separated by a colon (:).

> ***Note 1:*** An asterisk attached to a register name indicates that this register can be set *only* from the command line or before the MM macro definitions are read by the formatter {2.4, 2.5}.
>
> ***Note 2:*** Paragraph 1.6 has notes on setting and referencing registers. Any register having a single-character name can be set from the command line {2.4, 2.5}. These are indicated by a dagger (†) in the following list.

A *†
: Handles preprinted forms and logo
{2.4} 0, [0:2]

Au
: Inhibits printing of author information
{6.3} 1, [0:1]

C *†
: Copy type (original, DRAFT, etc.)
{2.4} 0 (Original), [0:4]

Cl
: Level of headings saved for table of contents
{4.4} 2, [0:7]

Cp
: Placement of list of figures, etc.
{10.1} 1 (on separate pages), [0:1]

D *†
: Debug flag
{2.4} 0, [0:1]

De
: Display eject register for floating dislays
{7.2} 0, [0:1]

Df — Display format register for floating displays
{7.2} 5, [0:5]

Ds — Static display pre- and post-space
{7.1} 1, [0:1]

E * † — Controls font of the Subject/Date/From fields
{2.4} 1 (**nroff**) 0 (**troff**), [0:1]

Ec — Equation counter, used by .EC macro
{7.5} 0, [0:?]
Incremented by one for each .EC call.

Ej — Page-ejection flag for headings
{4.2.2.1} 0 (no eject), [0:7]

Eq — Equation label placement
{7.4} 0 (right-adjusted), [0:1]

Ex — Exhibit counter, used by .EX macro
{7.5} 0, [0:?]
Incremented by one for each .EX call.

Fg — Figure counter, used by .FG macro
{7.5} 0, [0:?]
Incremented by one for each .FG call.

Fs — Footnote space (i.e., spacing between footnotes)
{8.4} 1, [0:?]

H1-H7 — Heading counters for levels 1 through 7
{4.2.2.5} 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)
{4.2.2.2} 2, [0:7]

Hc — Heading centering level for .H and .HU
{4.2.2.3} 0 (no centered headings), [0:7]

Hi	Heading temporary indent (after .H and .HU) {4.2.2.2} 1 (indent as paragraph), [0:2]
Hs	Heading space level (after .H and .HU) {4.2.2.2} 2 (space only after .H 1 and .H 2), [0:7]
Ht	Heading type (for .H: single or concatenated numbers) {4.2.2.5} 0 (concatenated numbers: 1.1.1, etc.), [0:1]
Hu	Heading level for unnumbered heading (.HU) {4.3} 2 (.HU at the same level as.H 2), [0:7]
Hy	Hyphenation control for body of document {3.4} 0 (automatic hyphenation off), [0:1]
L *†	Length of page {2.4} 66, [20:?] (11i, [2i:?] in **troff** formatter)
Le	List of equations {7.6} 0 (list not produced) [0:1]
Lf	List of figures {7.6} 1 (list produced) [0:1]
Li	List indent {5.1.1.1} 6 (**nroff**) 5 (**troff**), [0:?]
Ls	List spacing between items by level {5.1.1.1} 6 (spacing between all levels) [0:6]
Lt	List of tables {7.6} 1 (list produced) [0:1]
Lx	List of exhibits {7.6} 1 (list produced) [0:1]
N *†	Numbering style {2.4} 0, [0:5]
Np	Numbering style for paragraphs {4.1.2} 0 (unnumbered) [0:1]

O *†
Offset of page
{2.4} .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
{10.1} 0 (lowercase Roman), [0:1]

Of
Figure caption style
{7.5} 0 (period separator), [0:1]

P†
Page number managed by MM
{2.4} 0, [0:?]

Pi
Paragraph indent
{4.1.1} 5 (**nroff**) 3 (**troff**), [0:?]

Ps
Paragraph spacing
{4.1.3} 1 (one blank space between paragraphs), [0:?]

Pt
Paragraph type
{4.1.1} 0 (paragraphs always left justified), [0:2]

Pv
"PRIVATE" header
{9.10} 0 (not printed), [0:2]

Rf
Reference counter, used by .RS macro
{11.1} 0, [0:?]
Incremented by one for each .RS call.

S *†
The **troff** formatter default point size
{2.4} 10, [6:36]

Si
Standard indent for displays
{7.1} 5 (**nroff**) 3 (**troff**), [0:?]

T *†
Type of **nroff** output device
{2.4} 0, [0:2]

Tb — Table counter, used by .TB macro
{7.5} 0, [0:?]
Incremented by one for each .TB call.

U * † — Underlining style (**nroff**) for .H and .HU
{2.4} 0 (continuous underline when possible), [0:1]

W * † — Width of page (line and title length)
{2.4} 6i, [10:1365]
(6i, [2i:7.54i] in the **troff** formatter)

19. MM and Formatter Error Messages

When processing text using the MM macro package, MM will report any errors it can detect. Since the macros are written in the basic formatter primitives, other errors may be found and reported by the formatter (**nroff, troff/otroff**). The next two sections contain descriptions of the error messages which may be received from MM or the formatter.

19.1 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 parts consist of a descriptive message usually beginning with a macro name. They are listed below in alphabetical order by macro name, each with a more complete explanation.

Check TL, AU, AS, AE, MT sequence

The correct order of macros at the start of a memorandum is shown in paragraph 6.1. Something has disturbed this order.

Check TL, AU, AS, AE, NS, NE, MT sequence

The correct order of macros at the start of a memorandum is shown in paragraph 6.1. Something has disturbed this order. (Occurs if the .AS 2 {6.5} macro was used.)

CS:cover sheet too long

Text of the cover sheet is too long to fit on one page. The abstract should be reduced or the indent of the abstract should be decreased {6.5}.

DE:no DS or DF active

A .DE macro has been encountered, but there has not been a previous .DS or .DF macro to match it.

DF:illegal inside TL or AS

Displays are not allowed in the title or abstract.

DF:missing DE

A .DF macro occurs within a display, i.e., a .DE macro has been omitted or mistyped.

DF:missing FE

A display starts inside a footnote. The likely cause is the omission (or misspelling) of a .FE macro to end a previous footnote.

DF:too many displays

More than 26 floating displays are active at once, i.e., have been accumulated but not yet output.

DS:illegal inside TL or AS

Displays are not allowed in the title or abstract.

DS:missing DE

A .DS macro occurs within a display, i.e., a .DE has been omitted or mistyped.

DS:missing FE

A display starts inside a footnote. The likely cause is the omission (or misspelling) of a .FE to end a previous footnote.

FE:no FS active

A .FE macro has been encountered with no previous .FS to match it.

FS:missing DE

A footnote starts inside a display, i.e., a .DS or .DF occurs without a matching .DE.

FS:missing FE

A previous .FS macro was not matched by a closing .FE, i.e., an attempt is being made to begin a footnote inside another one.

H:bad arg:*value*

The first argument to the .H macro must be a single digit from one to seven, but *value* has been supplied instead.

H:missing arg

The .H macro needs at least one argument.

H:missing DE

A heading macro (.H or .HU) occurs inside a display.

H:missing FE

A heading macro (.H or .HU) occurs inside a footnote.

HU:missing arg

The .HU macro needs one argument.

LB:missing arg(s)

The .LB macro requires at least four arguments.

LB:too many nested lists

Another list was started when there were already six active lists.

LE:mismatched

The .LE macro has occurred without a previous .LB or other list-initialization macro {5.1.1}. This is not a fatal error. The message is issued because there exists some problem 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.

ML:missing arg

The .ML macro requires at least one argument.

ND:missing arg

The .ND macro requires one argument.

RF:no RS active

The .RF macro has been encountered with no previous .RS to match it.

RP:missing RF

A previous .RS macro was not matched by a closing .RF.

RS:missing RF

A previous .RS macro was not matched by a closing .RF.

S:bad arg:value

The incorrect argument *value* has been given for the .S macro {12.10}.

SA:bad arg:value

The argument to the .SA macro (if any) must be either 0 or 1. The incorrect argument is shown as *value*.

SG:missing DE

The .SG macro occurred inside a display.

SG:missing FE

The .SG macro occurred inside a footnote.

SG:no authors

The .SG macro occurred without any previous .AU macro(s).

VL:missing arg

The .VL macro requires at least one argument.

WC:unknown option

An incorrect argument has been given to the .WC macro {12.5}.

19.2 Formatter Error Messages

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

Cannot do ev

Caused by:

1. Setting a page width that is negative or extremely short
2. Setting a page length that is negative or extremely short
3. Reprocessing a macro package (e.g., performing a **.so** request on a macro package that was already requested on the command line)
4. Requesting the **troff** formatter -sl option on a document that is longer than ten pages.

Cannot execute *filename*

Given by the **.!** request if the *filename* is not found.

Cannot open *filename*

Indicates one of the files in the list of files to be processed cannot be opened.

Exception word list full

Indicates too many words have been specified in the hyphenation exception list (via **.hw** requests).

Line overflow

Indicates output line being generated was too long for the formatter line buffer capacity. The excess was discarded. Likely causes for this message are very long lines or words generated through the misuse of \c of the **.cu** request, or very long equations produced by **eqn/neqn**.

Nonexistent font type

Indicates a request has been made to mount an unknown font.

Nonexistent macro file

Indicates the requested macro package does not exist.

Nonexistent terminal type

Indicates the terminal options refer to an unknown terminal type.

Out of temp file space

Indicates additional temporary space for macro definitions, diversions, etc. cannot be allocated. This message often occurs because of unclosed diversions (missing .FE or .DE), unclosed macro definitions (e.g., missing " .."), or a huge table of contents.

Too many page numbers

Indicates the list of pages specified to the **–o** formatter option is too long.

Too many number registers

Indicates the pool of number register names is full. Unneeded registers can be deleted by using the **.rr** request.

Too many string/macro names

Indicates the pool of string and macro names is full. Unneeded strings and 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**.

Chapter 3

SROFF/MM USER GUIDE

PAGE

Chapter 3

SROFF/MM USER GUIDE

INTRODUCTION

This chapter is a user's guide for preparing documents with **sroff**/MM, a macro package for use with the **sroff** text formatter.

Styled after the Memorandum Macros (Chapter 2) for **nroff** and **troff**, the **sroff**/MM package offers a practical subset of the larger **nroff**/MM package plus a few formatting commands of its own.

Although not as extensive as the **nroff** version of MM, this **sroff** version provides enough macrocommands for most formatting applications. Naturally, this package includes macros for standard Bell Laboratories documentation styles, such as internal memoranda, released paper, and external letter. And there are the usual macros for the main body of text: paragraphs, section headings, itemized lists, footnotes, displays, etc. **Sroff**/MM does not provide facilities for automatic reference and table of contents processing. In return for its lack of capability vis-a-vis **nroff**, **sroff** will reward you tenfold in efficiency and response time.

In most cases, an sroff/MM macro shares the same name and argument list with its **nroff**/MM counterpart. Macros for paragraphs, headings, itemized lists, and document formats behave in much the same way as with **nroff**/MM. In addition, there are a few specialized macros (e.g. multiple column output) that are not available with **nroff**/MM.

Users should not be misled by intended similarities between the two macro packages; the text formatters (**sroff**, **nroff**, and **troff/otroff**) that use them are different in many respects. **Sroff** does not share the escape conventions and macro definition facilities of **nroff** and **troff**, nor does it have the luxury of **if-else** conditional requests or trap mechanisms. Furthermore, **sroff** cannot be used with the **tbl** table preprocessor or the **neqn/eqn** equation formatter.

With the UNIX operating system, users can use the **checkmm** command to scan the contents of input file(s) for errors in the use of Memorandum Macros. **Checkmm** was originally designed for use with **nroff**/MM, but works just as well with **sroff**/MM. This error-checking program checks for missing or unbalanced macro pairs (**.DS/.DE, .FS/.FE**), and improper usage of other MM macros; appropriate error messages are printed at the user's terminal.

For the occasions when you wish to convert a **nroff**/MM source file into a **sroff**/MM file, or vice versa, there is a command called **mmlint**. **MMlint** reads the source file(s) of the input document and reports the document changes required to convert the document into the specified text formatter.

Users who are familiar with **nroff**/MM may use the section at the end of this chapter *(SYNOPSIS OF SROFF/MM MACROS)* as a guide in preparing their documents. Be sure to check the argument list and default values for possible incompatibilities. At the very least, experienced **nroff**/MM users should read the next section before using **sroff**/MM.

The *SROFF/MM NUMBER REGISTERS* section contains a list of number registers used to control formatting styles. Users may change the values in these registers if the default values are not acceptable.

In Sections 3 through 12, macros are presented, more or less, in the order that they would be used. Macros used to produce Bell Laboratories document formats are discussed in Section 13. The discussion of each macrocommand begins with a synopsis of the macro call. Each macro is displayed with a generic list of arguments; optional arguments are enclosed in square brackets.

SROFF/MM REFERENCE GUIDE

1. Introduction to Macros

What is a macro? In **sroff**, a macro is a call to an executable text register containing a set of instructions to perform a specific formatting task. The text register has a one- or two-character name and is invoked as a request followed by an optional list of arguments. The instructions consist of **sroff** formatting requests intermixed with text fragments and parameters for argument substitution.

2. Macro Usage

2.1 Arguments and Quoting

Macros sometimes collect information by means of arguments. A list of one or more arguments is entered on the same line as the macro; each argument is separated by a space. For example,

```
.EX first second third fourth fifth
```

invokes a macro, **.EX**, with five arguments; each argument is an additional piece of information. Each macro may accept up to nine arguments.

If the text of an argument contains a space, the entire string must be enclosed in quotes (e.g. " string of text"). In this case, quoting is necessary to ensure that the text string is treated as a single argument.

A null argument is a pair of quotes with no text between them (" "). The null argument is used to skip or blank out an argument field.

To include a quote as part of the argument text, use two quotes. If the printable quote is the first character in the string, then the entire string must be enclosed in quotes.

2.2 Unpaddable Space Character

Sroff regards a space in the input text as a likely place to break an output line or to add extra space to fill an output line. An unpaddable space character may be used instead of a space to suppress breaking or padding at a given point.

In **sroff**/MM, the tilde (~) is defined as the unpaddable space character. On input, the tilde may be used instead of a space to ensure that two words are kept together on a line. On output, the tilde is translated into a single space.

To obtain a printable tilde on output, the **sroff** request

```
.tr~~
```

must be used before the tilde is to appear. This translates the tilde character into itself. After the line containing the tilde has been printed,

```
.tr~
```

restores the tilde as the unpaddable space character.

2.3 The Insertion Character and Predefined Registers

Information about the current date, time, and page number are available as predefined registers in **sroff**. These register names and their corresponding values are:

Register	*Contains*
%	current page number on output
(amon)	name of current month (e.g. January, February, March)
(mon)	number of current month of year (1-12)
(day)	date of current day of month (1-31)
(year)	last two digits of current year
(hour)	hour (0-23)
(min)	minute (0-59)
(sec)	second (0-59)

The contents of a register may be inserted in text by preceding the register name with an insertion character. **Sroff**/MM uses the circumflex (^) as the insertion character. For example,

```
Today is ^(amon) ^(day), 19^(year).
```

produces

Today is November 15, 1982.

The page number register (%) should not be preceded by the insertion character if used as part of the page header or page footer. For example, the macrocommand

```
.PF " '^(mon)/^(day)/^(year)"Page %'"
```

prints the current date and page number on the bottom of every page as shown below.

(The page footer macro, **.PF**, is discussed in Section 5.2.)

To obtain a printable circumflex, the **sroff** request

.ic

must be used before the circumflex is to appear. After the line containing the circumflex has been printed,

.ic^^

reinstates the circumflex as the insertion character. Each call to an **sroff**/MM macro automatically restores the circumflex as the insertion character.

2.4 Safe Sroff Requests

Sroff/MM provides the most commonly used formatting facilities. Additional tasks, like arranging tabular data (**.ta**) or changing pagination (**.bp, .pa, .sk**) can be performed with **sroff** formatting requests. Not all **sroff** requests should be used with *mm,* and only some requests can be used interchangeably with the **nroff/troff** formatters.

The following **sroff** requests are safe to use with *mm* and are compatible with **nroff/troff**:

Request	*Action*
.af *R F*	assign format *F* to number register *R*
.br	break output line
.bp	begin new page
.ce *N*	center next *N* lines
.fi	fill output lines
.hy *N*	turn hyphenation on (*N*=1) and off (*N*=0)
.in *%N*	indent *N* spaces

Request	*Action*
.ls *N*	*N* line spacing
.ne *N*	need *N* lines on page
.nf	do not fill output lines
.so *file*	insert source *file*
.sp *N*	space *N* lines
.ta *N N ...*	set tabs at *N*
.ti *%N*	indent only next line *N* spaces
.tr *abcd...*	translate *a* into *b*, *c* into *d*, etc.

The following **sroff** requests are not compatible with **nroff/troff**, but may be a useful addition to **sroff**/MM:

Request	*Action*
.li *N*	take next *N* lines literally
.lv *N*	leave *N* consecutive blank lines
.oc *C*	overstrike character is *C*
.pa *N*	begin new page with page number *N*
.sk *N*	skip at next page to page number *N*
.uc *C*	underline character is *C*
.us *N*	underscore all characters in next *N* lines
.ws *N*	suppress *N*-line leading widows

For additional information on these and other **sroff** requests, see *Text Formatters Reference*. .

Warning: Other sroff requests should be used with care, as they may conflict with the mm macros. Undefined requests will be printed as plain text.

3. Number Registers

.nr *name* [*value*]

The **.nr** macro assigns a value to a register that is used to control formatting style. The first argument to **.nr** is the name of the register and is traditionally one or two characters in length. The second argument to **.nr** is the integer value assigned to the register. If no value is given, that number register is set to zero.

Number registers are used internally by the macros to flag information and to do basic arithmetic such as incrementing and resetting counters. For example, section headings are automatically numbered through the use of such registers.

Some number registers are preset by **sroff**/MM to provide default formatting information, but the user may change a default value with the **.nr** macro. For example, the number register *Pi* contains the number of spaces used to indent the first line of an indented paragraph. Initially, *mm* sets *Pi* to 5, so the first line of all indented paragraphs is indented 5 spaces. If the user prefers an indentation of 3 spaces, the command

```
.nr Pi 3
```

changes the amount of paragraph indentation to 3 spaces.

Register values are usually set in the beginning of the document but may be changed anywhere in the document. The new values are used from that point on or until the register is changed again.

Actually, **.nr** is an **sroff** command and is synonymous with the **.an** command to set number register values. However, **sroff** syntax requires that register names consisting of more than one character be enclosed in parentheses. Therefore, **.nr** has been redefined as a

macro in *mm* to allow for compatibility with the **nroff** and **troff** text formatters. To increment or decrement the value in a number register, use the **sroff** request

.an (*name*) o*N*

The *SROFF/MM NUMBER REGISTERS* section contains a list of useful number registers. Users may change the value of these registers if the default values are not acceptable.

Two-letter register names of the form *aA*, where *a* and *A* are any lower- and upper-case alphabet, respectively, may be used freely without conflicting with **sroff**/MM.

4. Overall Text Format

4.1 Right Margin Justification

.SA [*mode*]

The **.SA** macro changes the right margin justification for the main body of text. By default, filled output lines are adjusted to produce an even right margin by filling extra blanks between words.

The justification *mode* (0 or 1) is given as an argument to **.SA**. If *mode* is 0, justification is turned off, producing a ragged right margin. If *mode* is 1 (or if **.SA** is invoked without an argument), justification is turned on, producing an even right margin.

4.2 Line Length, Page Length, and Page Offset

To change the default line length, page length, or page offset, reset the appropriate number register *and* invoke the corresponding **sroff** request:

		Default Values	
Format	**Terminal**	**Register**	**Request**
line length	65	W	.ll
page length	66	L	.pl
page offset	10	O	.po

For example,

```
.nr W 96
.ll 96
```

produces a 96-character line length.

5. Page Headers and Footers

Each page of straight text has a seven-line margin at the top and bottom. The margin at the top of the page consists of three initial blank lines, two header lines, and two more blank lines before the page text begins. The margin at the bottom of the page consists of two blank lines, two footer lines, and three remaining blank lines.

The header and footer lines may be blank or they may contain text, which is defined as a three-part title in the following format:

A delimiter is used to separate the parts of the title. Normally, the apostrophe (') is used as the delimiter, although any character will do, as long as it does not appear in the title text. Text between the first and second delimiter is printed at the left margin, text between the second and third delimiter is centered on the line, and text between the third and fourth delimiter is aligned at the right margin. Any or all of the three parts of the header or footer title may be blank.

The percent character (%) may be used in header and footer titles to obtain the current page number on output (see Section 2.3, *The Insertion Character and Predefined Registers*).

5.1 Page Headers

.PH ['*left*'*center*'*right*']
.OH ['*left*'*center*'*right*']
.EH ['*left*'*center*'*right*']

The **.PH** macro specifies the header that is to appear at the top of every page. The header text is given as an argument to **.PH** and is printed on the fourth line of the top-of-page margin.

The **.OH** macro specifies the header that is to appear at the top of every odd-numbered page. The **.EH** macro specifies the header that is to appear at the top of every even-numbered page. Text for the

odd-page header or the even-page header is given as an argument to **.OH** or **.EH**, respectively. Odd- and even-page headers are printed on the fifth line of the top-of-page margin, below the regular page header (if any).

By default, the current page number (centered and surrounded by dashes) is used as the header for every page; odd- and even-page headers are blank. All page headers are suppressed on the title page of internal memoranda, released papers, and external letters.

If **.PH**, **.OH**, or **.EH** are invoked without an argument, the respective header line is left blank.

5.2 Page Footers

.PF ['*left*'*center*'*right*']
.OF ['*left*'*center*'*right*']
.EF ['*left*'*center*'*right*']

The **.PF** macro specifies the footer that is to appear at the bottom of every page. The footer text is given as an argument to **.PF** and is printed on the fourth line of the bottom-of-page margin.

The **.OF** macro specifies the footer that is to appear at the bottom of every odd-numbered page. The **.EF** macro specifies the footer that is to appear at the bottom of every even-numbered page. Text for the odd-page footer or the even-page footer is given as an argument to **.OF** or **.EF**, respectively. Odd- and even-page footers are printed on the third line of the bottom-of-page margin, above the regular page footer (if any).

If **.PF**, **.OF**, or **.EF** are invoked without an argument, the respective footer line is left blank. By default, all page footers are blank.

6. Paragraphs

.P [*type*]

The **.P** macro marks the beginning of a paragraph. There are two types of paragraphs: left-adjusted paragraphs and indented paragraphs. In a left-adjusted paragraph, the text is blocked at the left margin; in an indented paragraph, the first line of text is indented from the left margin.

By default, paragraphs are left-adjusted. The number register *Pt* contains the default paragraph type. Legal values for *Pt* are 0 and 1: if *Pt* is set to 0, paragraphs are left-adjusted; if *Pt* is set to 1, paragraphs are indented.

The style of a single paragraph may be changed by specifying the desired paragraph type as the first argument to **.P**. This overrides the default paragraph type for that paragraph only. The command "**.P 1**" may be used to force a paragraph to be indented.

The number register *Pi* contains the amount of indentation used for indented paragraphs. By default, paragraphs are indented 5 spaces. The command

```
.nr Pi 10
```

changes the amount of paragraph indentation to 10 spaces.

The number register *Ps* contains the amount of space preceding paragraphs. By default, *Ps* is set to 1, producing one blank line before each paragraph. For example, in double space mode,

```
.nr Ps 0
```

could be used to suppress the extra blank space between paragraphs.

7. Section Headings

There are three kinds of section headings: sequentially numbered headings, unnumbered headings, and centered headings.

7.1 Numbered Headings

.H *level* [*heading_text*]

The **.H** macro can be used to obtain up to seven levels (sub-sections) of automatically numbered section headings. The first argument to **.H** specifies the heading level (1-7) and the second argument is the heading text. **.H** increments the counter for the specified heading level and generates current section numbers up to and including that level. The number registers *H1* through *H7* contain the current section numbers for heading levels 1 through 7, respectively.

.H produces the following format (spacing and print type) for heading levels 1, 2, and 3 through 7:

Macro call	*Format*
.H 1 " HEADING TEXT"	[2 blank lines] **#. HEADING TEXT** [**break**]
.H 2 " Heading Text"	[1 blank line] **#.#. Heading Text** [**break**]
.H 3 " Heading Text"	[1 blank line] *#.#.#. Heading Text* [No break occurs. Heading is immediately followed by text on the same line.]

The **.H** macro does not generate a blank line after headings. Therefore, **.sp** or any macro that normally generates a space (like **.P**) should follow first- and second-level headings; this is unnecessary (and often undesirable) in double space mode.

Warning: A .P macro must follow third- through seventh-level headings but will not cause a break after the heading text.

7.2 Unnumbered Headings

.HU [*heading_text*]

The **.HU** macro produces an unnumbered section heading. The heading text is supplied as an argument to **.HU**. Unnumbered headings are in the same format as major first-level headings: the heading is preceded by two blank lines and followed by a break, and heading text is set in bold type.

The **.HU** macro does not generate a blank line after the heading. Therefore, **.sp** or any macro that normally generates a space (like **.P**) should follow **.HU**. This is unnecessary (and often undesirable) in double space mode.

7.3 Centered Headings

Centered headings may be obtained by changing the value in the number register *Hc*. By default, *Hc* is set to 0, producing no centered headings. If *Hc* is set to 1, only major first-level headings are centered. If *Hc* is set to 2, both first- and second-level headings as well as unnumbered headings are centered. If *Hc* is set to 3, only unnumbered headings are centered.

8. Itemized Lists

An itemized list is a set of one or more blocked paragraphs which are indented from the left margin with a label to the left of each text block. There are six kinds of itemized lists: automatic lists, reference lists, dash lists, bullet lists, marked lists, and variable lists.

Three different macros must be used, in the following order, to generate an itemized list:

1. A list-initialization macro (**.AL, .RL, .DL, .BL, .ML,** or **.VL**) which specifies the desired list type and the overall appearance of the list.

2. One or more list-item macros (**.LI**) which mark the beginning of each item in the list.

3. A list-end macro (**.LE**) which marks the end of the list.

(The above numbered list of three items is an example of an automatic list.)

8.1 List Initialization

A list-initialization macro is used to begin a list and to specify the type of list being generated. All lists share the same basic format, but each list type serves a different purpose and has a different labeling style.

Each type of list has its own default indentation. The amount of indentation can be changed by specifying the desired number of spaces as the *text_indent* argument to the list-initialization macro.

Normally, one blank line separates the items in a list. This can be suppressed by specifying the number '1' as the last argument to the list-initialization macro.

8.1.1 Automatic List

.AL [*format_type*] [*text_indent*] [1]

The **.AL** macro begins an automatically numbered or alphabetized list. By default, list items are sequentially labeled with an arabic numeral followed by a dot (.). A different sequence format may be obtained by specifying one of the following format types as the first argument to **.AL**.

Type	*Sequence Format*
A	upper-case alphabet
a	lower-case alphabet
I	upper-case roman numerals
i	lower-case roman numerals
1	arabic numerals (default)
01,001,...	arabic numerals with corresponding number of leading zeros

Text blocks are indented 5 spaces unless otherwise specified.

8.1.2 Reference List

.RL [*text_indent*] [1]

The **.RL** macro begins an automatically numbered reference list. Each item in the list is sequentially labeled with an arabic numeral enclosed in square brackets.

Text blocks are indented 6 spaces unless otherwise specified.

8.1.3 Dash and Bullet Lists

.DL [*text_indent*] [1]
.BL [*text_indent*] [1]

The **.DL** and **.BL** macros begin dash and bullet lists, respectively. With the **.DL** macro, each item in the list is labeled with a dash (-); with the **.BL** macro, each item in the list is labeled with a bullet. On teletypewriter terminals, the bullet is simulated by overstriking the characters 'O' and '+'.

Text blocks in both dash and bullet lists are indented 2 spaces unless otherwise specified.

8.1.4 Marked List

.ML *label* [*text_indent*] [1]

The **.ML** macro begins a marked list. The marked list is similar to the dash and bullet lists in that the same label is used to mark each item in the list. However, in a marked list the user *must* supply the label. A label may consist of a single character or a string of characters. If no label is given, the list items will be unmarked.

Initially, text blocks are indented 5 spaces. Because the label may be of arbitrary length, a more appropriate indent should probably be specified.

8.1.5 Variable List

.VL [*text_indent*] [*mark_indent*] [1]

The **.VL** macro begins a variable list. This type of list is used when each list item is to be marked with a different label (as in a glossary of terms). In this case, the label is supplied as an argument to each list-item macro (**.LI**), and may consist of characters, words, or phrases.

If no label is given, the first line of the text block begins at the current margin, thereby producing a hanging indent as seen here.

Text blocks are indented 8 spaces unless otherwise specified.

In a variable list, the label itself may be indented by specifying the number of spaces as the *mark_indent* argument to **.VL**; otherwise, the label is printed at the current margin as in the other list types.

8.2 List Item

.LI [*label*]

The **.LI** macro marks the beginning of each item in a list, followed by the text for that list item. By default, **.LI** generates one blank line before each item; this can be suppressed by specifying the number '1' as the last argument to the list-initialization macro.

With the exception of variable lists, each item in a list is automatically marked with the label supplied by the list-initialization macro. In a variable list, the label must be supplied as the first argument to **.LI** for each item in the list. If no argument is given, a hanging indent will be generated for that item.

8.3 List End

.LE [1]

The **.LE** macro marks the end of an itemized list. List indentation is turned off and the left margin is reset to where it was before that list began. Normally, **.LE** does not generate a blank line at the end of a list; it is assumed that most lists will be followed by a macro that generates its own space (such as **.P** or **.H**). If a list is followed by running text, a blank line between the list and text can be obtained by specifying the number '1' as an argument to **.LE**.

8.4 Nested Lists

It is possible to produce a list within a list (nested list) or even a list with several levels of nested lists (multiple nested list). The basic structure of an ordinary list applies to nested lists as well: each list must begin with one of the list-initialization macros, must contain at least one list item (marked by **.LI**), and must end with the list-end macro (**.LE**). A sub-list may begin anywhere after the first item in the current list, and must end before either the next item or the end of the current list. Each sub-list begins at the current list indent, and reverts to the previous indent when ended. The left margin is shifted to the right with each list-initialization macro, and shifted to the left with each **.LE**.

The sample input in Figure 3-1 was used to generate the multiple nested list in Figure 3-2.

Although there is no limit to the depth of a nested list, anything beyond a fourth nested level may be unreasonable. In fact, depending on the types of lists used, a sixth nested level may fall off the page. The more levels used, the more complicated the input will be. Users should be careful in organizing complex lists and should make certain that there is a **.LE** for every list-initialization macro used. (**Checkmm** would be extremely helpful here.)

```
The following multiple nested list begins at the current
left margin:
.AL
.LI
First item in automatically numbered list.
.LI
Second item in automatically numbered list.
.AL a
.LI
First item in alphabetized sub-list.
Notice how the left margin is indented to the right.
.BL
.LI
Item in a bullet-type sub-list.
Again, the left margin is reset.
.LI
Another item in the same sub-list.
.LI
Yet another item in this sub-list.
.LE
.LI
Second item in the alphabetized sub-list.
Notice how the left margin was reset to the
previous indent.
.DL
.LI
Item in a dash-type sub-list.
Again, the left margin is reset to the right.
.LI
Another item in the same sub-list.
.LE
.LE
.LI
Third (and last) item in automatically numbered list.
Notice how the left margin was reset to the proper
list level.
.LE 1
Now the left margin is restored to its original
state before the outermost list began.
```

Figure 3-1. Sample Input For a Multiple Nested List

The following multiple nested list begins at the current left margin:

1. First item in automatically numbered list.

2. Second item in automatically numbered list.

   a. First item in alphabetized sub-list. Notice how the left margin is indented to the right.

      • Item in a bullet-type sub-list. Again, the left margin is reset.

      • Another item in the same sub-list.

      • Yet another item in this sub-list.

   b. Second item in the alphabetized sub-list. Notice how the left margin was reset to the previous indent.

      — Item in a dash-type sub-list. Again, the left margin is reset to the right.

      — Another item in the same sub-list.

3. Third (and last) item in automatically numbered list. Notice how the left margin was reset to the proper list level.

Now the left margin is restored to its original state before the outermost list began.

Figure 3-2. Sample Output For a Multiple Nested List

9. Footnotes

.FS *label* [*text_indent*]
.FE

The **.FS** and **.FE** macros mark the beginning and end of the footnote text.

The footnote macros and text should be typed on the next line following the word being footnoted. The footnote label should be typed at the end of the footnoted word and given as an argument to **.FS**. The following example shows the input used to produce the rest of this sentence and the footnote* at the bottom of this page.

```
and the footnote*
.FS *
Sroff collects the footnote text and saves it
for the bottom of the current page.
A one-inch line separates the footnote from
the main body of text.
.FE
at the bottom of this page.
```

By default, the footnote text is indented 2 spaces from the left margin. The amount of indentation can be changed by specifying the desired number of spaces as the second argument to **.FS**.

---

* **Sroff** collects the footnote text and saves it for the bottom of the current page. A one-inch line separates the footnote from the main body of text.

10. Displays

.DS [*format*] [*mode*] [*r_indent*]
.DE [1]

The **.DS** and **.DE** macros delimit text that is to be kept together on a page. The text between **.DS** and **.DE** may consist of one or more lines of text and may contain simple **sroff** formatting commands (such as **.sp**, **.ce**, and **.ul**). The text is formatted and displayed as a single block; on output, one blank line precedes and follows the display.

If possible, a display is printed where it is invoked. If a display does not fit on the current page, then the *remainder of the page is left blank* and the display is printed at the top of the next page.

If **.DS** is invoked with no arguments, the text lines are neither filled nor adjusted, and the entire display is blocked at the current left margin. The *format*, *mode*, and *r_indent* arguments to **.DS** may be used to alter the appearance of a display. For historical reasons, *format* and *mode* may be an integer or letter.

The first argument to **.DS** controls the format of the display. If *format* is 0 or L, the entire display is printed at the current left margin. (This is the default format.) If *format* is 1 or I, the entire display is indented from the current left margin. If *format* is 2 or C, each line of the display is centered individually. (Unfortunately, there is no way to automatically center an entire display as a single block.)

The number register *Si* contains the amount of indentation used for indented displays. Unless otherwise specified, displays with format 1 or I are indented 5 spaces from the current left margin.

The second argument to **.DS** controls line filling and adjustment. If *mode* is 0 or N, the display is processed in no-fill mode and each text line is printed exactly as typed. If *mode* is 1 or F, the display is processed in fill mode and output lines are filled to current available line length. By default, displays are processed in no-fill mode.

Note: The **sroff** requests **.nf** and **.fi** have no effect in displays with *mode* 1 or F.

The third argument to **.DS** specifies the amount of indentation from the right margin. By default, text is not indented from the right margin.

To display a block of text with matching left- and right-hand indents, use

```
.nr Si n
.DS I F n
```

where *n* is the desired amount of indentation. For example,

```
.nr Si 4
.DS I 1 4
.ti +3
" To retire is not to flee, and there is no wisdom
waiting when danger outweighs hope, and it is
the part of wise men to preserve themselves
today for tomorrow, and not risk all in one day."
.sp
.ti +20
- Cervantes
.ul
(Don Quixote)
.DE
```

produces the following quotation format:

> "To retire is not to flee, and there is no wisdom waiting when danger outweighs hope, and it is the part of wise men to preserve themselves today for tomorrow, and not risk all in one day."
>
> - Cervantes (Don Quixote)

Normally, **.DE** generates a blank line at the end of the display; it is assumed that most displays will be followed by running text. If a display is followed by a macro that generates its own spaces (such as **.P** or **.H**), the extra blank line following the display can be suppressed by specifying the number '1' as an argument to **.DE**.

Warning: Displays cannot be used in multiple column format, and footnotes cannot be used within displays.

11. Multiple Columns

Text format can change from single-column (default) to double- or multi-column. Double and multiple (3 to 10) columns of text are automatically balanced on partially filled pages; a change in column format also produces even columns of text. Switching back and forth between column formats does not cause page breaks.

11.1 Double- and Single-Column Format

.2C
.1C

The **.2C** macro generates double-column output. All lines of text following **.2C** are processed in two-column mode. The columns are balanced, producing two even columns of text.

The **.1C** macro generates single-column output, and is used to turn off the effects of **.2C**. Double-column text is balanced before **.1C** reverts to single-column processing.

11.2 Multi-Column Format

.MC [*columns*]

The **.MC** macro generates multi-column output. The number of columns is given as an argument to **.MC**; output may be from 1 to 10 columns wide. If **.MC** is invoked without an argument, text is processed in single-column mode. (**.1C** and **.2C** are merely extensions of the **.MC** macro.)

Multi-column format is helpful in displaying a blocked list of data or word items. For example,

```
.MC 5
.nf
one
two
three
 ...
nineteen
.fi
.MC 1
```

prints each item separately (**.nf**) and successively lists the items in five columns:

```
one         five        nine        thirteen    seventeen
two         six         ten         fourteen    eighteen
three       seven       eleven      fifteen     nineteen
four        eight       twelve      sixteen
```

Notice how the columns are balanced. To display the same information row-wise, use tabs to separate the items on each line.

12. Emboldening and Italics

.B [*word*]
.I [*word*]
.R

Words or lines of text are emphasized by using a different font (or print) type. The **.B** macro emboldens text by overstriking each character once. The **.I** macro emphasizes text with italics or underscores. All characters are underscored on teletypewriter terminals.

If a word or phrase is given as an argument to **.B**, only that word is emboldened. If **.B** is invoked without an argument, all following lines of text are emboldened until a **.R** or **.I** macro is used.

Similarly, if a word or phrase is given as an argument to **.I**, only that word is italicized (or underscored). If **.I** is invoked without an argument, all following lines of text are italicized until a **.R** or **.B** macro is used.

For example,

```
Only this
.B word
is emboldened.
.B
But all of this text is
set in bold type.
.I
And all of this text is set
in italic type.
.R
Now text is back to the
regular type, except
for this
.I italicized
word.
```

produces

Only this **word** is emboldened. **But all of this text is set in bold type.** *And all of this text is set in italic type.* Now text is back to the regular type, except for this *italicized* word.

The **.R** macro turns off the effect of **.B** and **.I** and restores the regular print type.

13. Documentation Styles

Sample input and output for available documentation styles is included in the *EXAMPLES OF INPUT AND OUTPUT* section at the end of this chapter.

13.1 Macros for the Beginning

This section deals with a set of macros used to format the title page of standard Bell Laboratories documents such as internal memoranda, released papers, and external letters. These macros may also be used by companies other than Bell Laboratories.

Title page information (for title, author, abstract, etc.) is typed in the same way for all document styles. The information collected by each title page macro is then used by the **.MT** macro to generate the proper format.

The title page macros (each of which will be discussed separately in the sections that follow) are used in the beginning of a document before the main body of text. The macros must be invoked in the following order:

date	**.ND** [*date*]
title	**.TL** [*charging*] [*filing*] [*#_lines*]
	one or more lines
	of title text
author	**.AU** *name* [*init*] [*loc*] [*dept*] [*ext*] [*room*]
affiliation	**.AA** [*name and address*] ...
TM or MF number	**.TM** [*number*] or **.MF** [*number*]
abstract start	**.AS** [*mode*] [*indent*] [*heading*]
	one or more lines
	of abstract text
abstract end	**.AE**
document style	**.MT** [*type*] [*addressee*]

The use of **.TL**, **.AU**, and **.MT** is mandatory in order to produce the proper document format; use of the other title page macros is optional. Other macros should not be used within this block.

13.1.1 Date

.ND [*date*]

The **.ND** macro changes the publication date normally printed on memoranda, released paper cover sheets, and external letters. If **.ND** is not used, **sroff**/MM prints the current date in the format **August 15, 1983**.

A fixed date may be specified as an argument to **.ND**. The new date will be printed as typed. If **.ND** is invoked without an argument, no date will be used and the date field will be left blank.

In memorandum-type documents, the date is printed on the title page after the word " date:" . In released paper format, the date appears only on the cover sheet (if any), and is printed at the left margin below the abstract. Since external letters are usually printed on company letterhead, the date appears directly below where the letterhead block would be.

13.1.2 Title

.TL [*charging_case*] [*filing_case*] [*#_of_lines*]

The **.TL** macro collects information for title processing. The title itself begins on the line after **.TL** and may consist of one or more lines of text.

The charging case number and filing case number (if any) are given as the first and second argument to **.TL**, respectively. Multiple case numbers should be separated by a comma and a space (,) and treated as a single argument, as in

```
.TL " 82100-100, 82108-200" " 39199-11, 39400-02"
```

Case numbers are used only for internal memoranda and are ignored in released papers and external letters. These may be left out for

documents outside Bell Laboratories (BTL) or documents that don't require them.

The third argument to **.TL** is the number of lines the title should have *on output.* This is only used for memorandum-style documents and can be omitted for released paper and external letter formats. By default, **.TL** assumes that the title for an internal memorandum will take up one line. If the user suspects that the " subject" information may consist of two or more output lines, it may be necessary to test-print the first page to get the right number.

In memorandum-style documents, the title is printed as a filled block of text in bold type and is indented after the word " subject" . If one or both case numbers are used, they will be printed on separate lines following the title. For example, as part of the input for an internal memorandum

```
.ND " February 22, 1982"
.TL 82108-200 39199-11 2
The Title of this Memorandum
is " An MM-Like Macro Package
for Sroff"
```

produces

```
subject: The Title of this Memorandum is                    date: February 22, 1982
         "An MM-Like Macro Package for Sroff"
         Charging Case 82108-200                            from:
         Filing Case 39199-11
```

(Notice the third argument to **.TL** reflects the number of lines used for the title *on output,* not on input, and does not include lines used for the charging and filing case.)

In a released paper, the title is centered and emboldened.

Normally, a title does not appear on an external letter and can be suppressed by omitting the title text altogether. If given, the title is printed as a filled block of text in the upper left side of the page.

13.1.3 Author

.AU *name* [*initials*] [*location*] [*department*] [*extension*] [*room*]

The **.AU** macro collects information about the author(s). Since **.AU** is also used to mark the end of the title text, it *must* be the first command after the title. If the document has more than one author, a separate **.AU** must be used to describe each author. Up to six authors are allowed in a document.

The **.AU** macro accepts up to six arguments. The first argument is the author's name, and is used on the title page and in the signature block at the end of the document. The author's initials are given as the second argument and are usually typed in upper-case letters for use in the signature block's reference line.

The author's location, department number, telephone extension, and room number are given as the third, fourth, fifth, and sixth arguments, respectively. They are used on the title page of internal memoranda; the location and department number are also used in the reference line.

The information collected by **.AU** is used in different ways depending on the document type. For example, in a memorandum-type document, the macro call

```
.AU " S. P. LeName" CL MH 12345 6789 1A-000
```

produces

```
from: S. P. LeName
      MH 12345
      1A-000 x6789
```

(Notice that **sroff**/MM automatically prefixes the extension number with **x**.) In addition, the name, initials, location, and department number are used by the **.SG** macro (see Section 13.2.2) to generate the signature block at the end of the memorandum.

In released paper format, only the author's name and location code are used and any other arguments to **.AU** are ignored. The author's name is centered and underscored below the title. The location code is used to select the corresponding BTL address, which is centered below the author's name. For example, the macro calls for a released paper with multiple authors

```
.TL
A Short Title
.AU " S. P. LeName" CL MH 12345 6789 1A-000
.AU " H. O. Employee" HOE HO
```

produce

A Short Title

S. P. LeName

Bell Laboratories
Murray Hill, New Jersey 07974

H. O. Employee

Bell Laboratories
Holmdel, New Jersey 07733

Addresses are provided for the following BTL location codes: AK, AL, CB, DR, HL, HO, HP, IH, IN, MH, MV, PY, RD, WB, and WH. (The company addresses for other BTL locations and for non-BTL authors are discussed in Section 13.1.4, *Author's Affiliation.*)

In external letter format, only the author's name and initials are used and the remaining arguments to **.AU** are ignored. The name and initials are used by the **.SG** macro (see Section 13.2.2) to generate the signature block at the end of the letter.

13.1.4 Author's Affiliation

.AA [*1st_line*] [*2nd_line*] [*3rd_line*]

The **.AA** macro, used only for released paper format, specifies the institution's name and address for non-BTL authors. It may also be used for authors from BTL locations for which an address is not automatically provided (i.e. ALF, FJ, NP, etc.).

.AA accepts up to three arguments, one for each line of the institution's address. For example,

```
.TL
A Physical Phenomenon
.AU " I. M. Here" IMH NP 11312
.AA " Bell Laboratories" " Neptune, New Jersey 07753"
.AU " U. R. There" URT
.AA " Department of Physics" " Princeton University" " Princeton, New Jersey 08540"
```

produces

A Physical Phenomenon

I. M. Here

Bell Laboratories
Neptune, New Jersey 07753

U. R. There

Department of Physics
Princeton University
Princeton, New Jersey 08540

Each **.AA** must follow the corresponding **.AU** for that author.

13.1.5 TM and MF Numbers

.TM [*number*]
.MF [*number*]

Filing numbers are usually assigned to Technical Memoranda and Memoranda for File by the department secretary. For Technical Memoranda, the TM number is given as an argument to **.TM**; for Memoranda for File, the MF number is given as an argument to **.MF**. If used, the TM or MF number is printed below the author information on the title page and cover sheet. Multiple TM or MF numbers should be separated by a space and treated as a single argument.

13.1.6 Abstract and Cover Sheet

.AS [*mode*] [*indent*] [*heading*]
.AE

The **.AS** macro marks the beginning of the abstract and the **.AE** macro marks its end. The abstract, which is sandwiched between **.AS** and **.AE**, may consist of one or more lines of text and may contain simple **sroff** formatting commands (such as **.sp**, **.ce**, and **.ul**).

The first argument to **.AS** determines whether the abstract appears on the cover sheet or on the title page or both. By default, *mode* is 0 and the abstract is printed only on the cover sheet. If *mode* is 1, the abstract appears only on the title page. If *mode* is 2, the abstract appears on both the cover sheet and the title page.

By default, the abstract text is printed with ordinary text margins. The abstract may be indented from both margins by specifying the number of spaces as the second argument to **.AS**. For example,

```
.AS 0 4
This is the text for a very short abstract.
It really doesn't say anything since it is only
.ul
filler
used for this " you type" /" you get"  example.
.sp
What you get here is just an abstract.
Normally, because the first argument to .AS is 0,
the abstract would appear only on the cover sheet,
rather than in the middle of the document as seen here.
.AE
```

produces

ABSTRACT

This is the text for a very short abstract. It really doesn't say anything since it is only filler used for this "you type"/"you get" example.

What you get here is just an abstract. Normally, because the first argument to .AS is 0, the abstract would appear only on the cover sheet, rather than in the middle of the document as seen here.

If the **.AS** and **.AE** macros are used, a centered heading (*ABSTRACT*) followed by a blank line and the abstract text is printed after the title/author information. On the title page, three blank lines separate the abstract from the main body of the document. An alternate heading may be given as the third argument to **.AS**, replacing the word "*ABSTRACT*".

The abstract macros are used in memorandum-type documents and released papers, and are ignored in external letter format. To obtain an abstract on a separate cover sheet, **.AS** must be invoked with *mode* 0 or 2.

13.1.7 Document Type

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

The **.MT** macro generates standard Bell Laboratories format for the cover sheet and title page of both internal and external correspondence. Available documentation styles include internal memoranda, released paper, and external letter formats.

The document type is given as the first argument to **.MT**. For internal correspondence, the following values for *type* generate memorandum format with the corresponding heading:

type	*memorandum heading*
0	
1	*MEMORANDUM FOR FILE*
2	*PROGRAMMER'S NOTES*
3	*ENGINEER'S NOTES*
" *string*"	*string*

The memorandum heading (if any) is centered four spaces below the " subject/date/from" block on the title page. If **.MT** is invoked without an argument (or if *type* is 0), a simple memorandum is generated. If *type* is a text string (e.g., " CONFERENCE NOTES", " MEMORANDUM FOR RECORD"), then that string is used as the memorandum heading.

For external correspondence, values for *type* and the corresponding document formats are:

type	*document format*
4	released paper
5	external letter

Sample input and output for available documentation styles is included in the *EXAMPLES OF INPUT AND OUTPUT* section at the end of this chapter.

Information used on the cover sheet and title page is supplied as arguments to the title page macros (**.ND**, **.TL**, **.AU**, etc.) and is used by **.MT** to generate the proper document format. **.MT** uses only the information needed for the specified document type; any irrelevant information is ignored. This can be used to your advantage: by including all information you need only change **.MT** *type* to obtain a different style.

Page header information is suppressed on the cover sheet and title page of all document styles. Unless otherwise specified, the default page header is used on the second and succeeding pages of the document.

For addressed memoranda and external letters, the name of the addressee may be given as the second argument to **.MT**. The addressee's name, followed by a dash and the current page number, will replace the page header for the second and succeeding pages of the document. The *addressee* argument is ignored in the released paper format.

13.2 Macros for the End

13.2.1 Formal Closing

.FC [*closing*]

The **.FC** macro generates the formal closing normally used in an external letter. If **.FC** is invoked without an argument, the closing

Yours very truly,

is used. A different closing (e.g. " Sincerely," , " Yours truly,") may be given as a quoted argument to **.FC**.

The formal closing is ignored in memorandum and released paper formats.

13.2.2 Signature Block

.SG [*typist's_initials*]

The **.SG** macro produces the signature block at the end of memorandum-type documents and external letters. One signature line is produced for each author. The author's name is printed on the right side of the page and is preceded by three blank lines for the author's signature. The reference data, taken from the first **.AU**, is printed at the left margin on the same line as the name of the last author.

The typist's initials are given as an argument to **.SG** and are appended to the reference data. In memorandum-type documents, the reference data includes the first author's location, department number, and initials, and the typist's initials. For example, given this author information at the beginning of the document

```
.AU " A. D. Strauss"  ADS MH 37842
.AU " C. G. Bellows"  CGB HO 37635
```

the command

```
.SG cyd
```

at the end of the document produces

```
                                        A. D. Strauss


MH-37842-ADS-cyd                        C. G. Bellows
```

In external letter format, the reference data includes the author's initials and the typist's initials separated by a colon (i.e. ADS:cyd).

If **.SG** is invoked without an argument, the reference data is omitted from the signature block.

The signature block is not used in released paper format, and the **.SG** macro is ignored.

13.2.3 Notations

.NS [*type*]
.NE

The **.NS** and **.NE** macros may be used to generate several types of notation lists at the end of a memorandum or letter. Typically, notations (i.e. " Copy to", " Att.") appear after the signature block and before the approval block (if any).

The first argument to **.NS** specifies the type of notation. If no argument is given, a " Copy to" notation is generated. The following table shows the notation types that are available:

type	*notation*
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

A separate **.NS** is used for each notation type. The notation list, if any (i.e. list of names, attachments, enclosures), follows the corresponding **.NS**. One **.NE** is used after the last notation to mark the end of the entire notation block.

Notations are printed at the current left margin. Each notation is preceded by one blank line and the notation list (if any) is printed exactly as typed. For example,

```
.NS 4
As above
.NS 1
C. V. Klein
J. P. Peters
A. G. Sellem
.NS " with att. 1 only"
J. D. Murphy
.NS 2
All Members Department 12345
.NE
```

produces

Atts.
As above

Copy (with att.) to
C. V. Klein
J. P. Peters
A. G. Sellem

Copy (with att. 1 only) to
J. D. Murphy

Copy (without att.) to
All Members Department 12345

The notation macros are used in memorandum-type documents and external letters, and are ignored in the released paper format.

13.2.4 Approval Block

.AV *name*

The **.AV** macro generates an approval block with lined space for the approver's signature and the date. The name of the approver is given as an argument to **.AV** and is printed below the signature line. For example,

```
.AV " O. K. Withme"
```

produces

```
APPROVED:


_____________________________________    _______________
O. K. Withme                             Date
```

Typically, approvals appear after the author's signature and notation list (if any). The approval block is used in external letters and memorandum-type documents; the **.AV** macro is ignored in released paper format.

14. Usage

With the UNIX operating system, users can use the **checkmm** command to scan the contents of the input file(s) for errors in the use of Memorandum Macros. This error-checking program checks for missing or unbalanced macro pairs (**.DS/.DE**, **.FS/.FE**), and improper usage of other MM macros; appropriate error messages are printed at the user's terminal.

Documents prepared with **sroff**/MM may be printed on any computer terminal or line printer. The command

sroff −**mm** *files*

generates output at the user's terminal. Documents may be printed at a remote line printer by piping the output through the appropriate filter.

15. Sroff vs. Nroff

With the *mm* macro package, the performance ratio of **sroff** vs. **nroff** is almost 3 to 1; without any macros, the ratio is more like 10 to 1. Since **sroff** (and **nroff**) must process all the text that is part of the macro package, the size of the package directly affects the speed of the process. For this reason, many formatting luxuries like table of contents processing and automatic referencing were not implemented in **sroff**/MM.

Remember, the command **mmlint** may be used to run compatibility checks on **nroff**/MM and **sroff**/MM source files. If you wish to see what changes are necessary to convert a **nroff**/MM source file into a form usable with **sroff**/MM text formatter requests, you enter:

mmlint -s *files*

If you wish to see what changes are necessary to convert a **sroff**/MM source file into a form usable with **nroff**/MM text formatter requests, you enter:

mmlint -n *files*

In either case, appropriate error messages are printed at the user's terminal indicating the action required for the specified conversion.

SYNOPSIS OF SROFF/MM MACROS

The following listing shows all the **sroff**/MM macros and their usage. Macro commands are listed, more or less, in the order that they would be used. Each macro is followed by its argument list. Optional arguments are enclosed in square brackets and default values are enclosed in parentheses. Legal parameter values are listed below the argument list.

MACRO FUNCTION	NAME	ARGUMENT LIST
set number register	**.nr**	*name* [*value*(0)]
margin justification	**.SA**	[*flag*(1)]
		flag: 1 = on 0 = off
page header	**.PH**	['*left*'*center*'*right*']
odd page header	**.OH**	['*left*'*center*'*right*']
even page header	**.EH**	['*left*'*center*'*right*']
page footer	**.PF**	['*left*'*center*'*right*']
odd page footer	**.OF**	['*left*'*center*'*right*']
even page footer	**.EF**	['*left*'*center*'*right*']
date	**.ND**	[*date*]
title	**.TL**	[*charging_case*] [*filing_case*] [*#_lines*(1)]
author	**.AU**	*name* [*initials*] [*loc.*] [*dept.*] [*ext.*] [*room*]
author's affiliation	**.AA**	[*1st_line_address*] [*2nd_line*] [*3rd_line*]
TM number	**.TM**	[*number*]
MF number	**.MF**	[*number*]

abstract start	**.AS**	[*flag*(0)] [*indent*(0)] [*heading*(ABSTRACT)] *flag*: 0 = cover sheet only 1 = title page only 2 = cover sheet and title page
abstract end	**.AE**	
document type	**.MT**	[*type*(0)] [*addressee*] *type*: 0 = no memo type printed 1 = MEMORANDUM FOR FILE 2 = PROGRAMMER'S NOTES 3 = ENGINEER'S NOTES 4 = released paper 5 = external letter " *string*" = *string* printed
paragraph	**.P**	[*type*(0)] *type*: 0 = blocked 1 = indented
numbered heading	**.H**	*level* [*heading_text*] *level*: 1 through 7
unnumbered heading	**.HU**	[*heading_text*]
automatic list	**.AL**	[*format*(1)] [*indent*(5)] [1] *format*: 1 = arabic A = upper-case alphabet a = lower-case alphabet I = upper-case roman i = lower-case roman 01,001,... = arabic with leading zeroes
reference list	**.RL**	[*indent*(6)] [1]
dash list	**.DL**	[*indent*(2)] [1]
bullet list	**.BL**	[*indent*(2)] [1]

marked list	**.ML**	*mark* [*indent*(5)] [1]
variable list	**.VL**	[*text indent*(8)] [*mark indent*(0)] [1]
list item	**.LI**	[*mark*]
list end	**.LE**	[1]
footnote start	**.FS**	*label* [*text_indent*]
footnote end	**.FE**	
display start	**.DS**	[*format*(0,L)] [*fill*(0,N)] [*right indent*(0)] *format*: 0 or L = left-adjusted 1 or I = indented 2 or C = centered *fill*: 0 or N = no fill 1 or F = fill
display end	**.DE**	[1]
single column	**.1C**	
double column	**.2C**	
multiple column	**.MC**	[*columns*(1)] *columns*: 1 thru 10
boldface type	**.B**	[*string*]
italic type	**.I**	[*string*]
regular type	**.R**	
formal closing	**.FC**	[*closing*(Sincerely,)]
signature	**.SG**	[*typist's_initials*]

notation start **.NS** [*type*(0)]

type: 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

notation end **.NE**

approval line **.AV** *name*

SROFF/MM NUMBER REGISTERS

The following number registers may be reset with the **.nr** command. Default values are initially set by **sroff**/MM.

Register Name	Default Value	Usage
H1	0	Level 1 heading number
H2	0	Level 2 heading number
H3	0	Level 3 heading number
H4	0	Level 4 heading number
H5	0	Level 5 heading number
H6	0	Level 6 heading number
H7	0	Level 7 heading number
Hc	0	Centered heading level
L	66	Page length*
O	10	Page offset*
Pi	5	Paragraph indent
Ps	1	Paragraph spacing
Pt	0	Paragraph type
Si	5	Display indent
W	65	Line length*

Note: Two-letter register names of the form *aA*, where *a* and *A* are any lower- and upper-case alphabet, respectively, may be used freely without conflicting with **sroff**/MM.

* Must invoke the corresponding **sroff** request for the change to take effect.

EXAMPLES OF INPUT AND OUTPUT

The following input was used to produce both the sample Memorandum for File (using **.MT 1**) *and* the sample Released Paper (using **.MT 4**) shown in the next two sections.

Sample Input:

```
.ND "December 7, 1981"
.TL 12345 54321
Preparing Documents with Sroff/MM
.AU "S. P. LeName" SPL MH 45231 6351 2F-120
.AU "C. Walker" CW MH 45231 3732 2F-122
.AA "University of Wisconsin" "Madison, Wisconsin  53701"
.MF 81-45231-09
.AS 0 5
This is the text for an abstract which will appear
only on the cover sheet, and will be indented five
spaces from both left and right margins.
.AE
.MT type
.H 1 "FIRST-LEVEL HEADING"
.P
The .P macro marks the beginning of a paragraph.
By default, paragraphs are blocked at the
left margin and preceded by a blank line.
.H 2 "Second-Level Heading"
.P
Both first- and second-level headings, as well as unnumbered
headings, are set apart from the text; section numbers and
heading text are emboldened.
.H 3 "Third-Level Heading"
.P
This is the first paragraph under the third-level heading.
The section number and heading text are italicized,
and the text of this paragraph immediately follows the
heading text.
.H 1 "DOCUMENTATION STYLES"
.P
The .MT macro controls the documentation style.
The same input was used to produce
both memorandum and released paper formats.
The macro call, .MT 1, was used to produce the sample
Memorandum for File; the macro call, .MT 4,
was used to produce the sample released paper.
.SG cl
.NS 4
Appendices 1-5
.NE
```

Sample Output: Memorandum for File (.MT 1)

subject: **Preparing Documents with Sroff/MM**
Charging Case 12345
Filing Case 54321

date: **December 7, 1981**

from: **S. P. LeName**
MH 45231
2F-120 x6351

C. Walker
MH 45231
2F-122 x3732

MF 81-45231-09

ABSTRACT

This is the text for an abstract which will appear only on the cover sheet, and will be indented five spaces from both left and right margins.

Sample Output: Memorandum For File (Contd)

subject: **Preparing Documents with Sroff/MM**
Charging Case 12345
Filing Case 54321

date: **December 7, 1981**

from: **S. P. LeName**
MH 45231
2F-120 x6351

C. Walker
MH 45231
2F-122 x3732

MF 81-45231-09

MEMORANDUM FOR FILE

1. FIRST-LEVEL HEADING

The .P macro marks the beginning of a paragraph. By default, paragraphs are blocked at the left margin and preceded by a blank line.

1.1 Second-Level Heading

Both the first- and second-level headings, as well as unnumbered headings, are set apart from the text; section numbers and heading text are emboldened.

1.1.1. Third-Level Heading This is the first paragraph under the third-level heading. The section number and heading text are italicized, and the text of this paragraph immediately follows the heading text.

2. DOCUMENTATION STYLES

The .MT macro controls the documentation style. The same input was used to produce both memorandum and released paper formats. The macro call, .MT 1, was used to produce the sample Memorandum for File; the macro call, .MT 4, was used to produce the sample released paper.

S. P. LeName

MH-12345-SPL-cl

C. Walker

Atts.
Appendices 1-5

Sample Output: Released Paper (.MT 4)

Preparing Documents with Sroff/MM

S. P. LeName

Bell Laboratories
Murray Hill, New Jersey 07974

C. Walker

University of Wisconsin
Madison, Wisconsin 53701

ABSTRACT

This is the text for an abstract which will appear only on the cover sheet, and will be indented five spaces from both left and right margins.

December 7, 1981

Sample Output: Released Paper (Contd)

Preparing Documents with Sroff/MM

S. P. LeName

Bell Laboratories
Murray Hill, New Jersey 07974

C. Walker

University of Wisconsin
Madison, Wisconsin 53701

1. FIRST-LEVEL HEADING

The .P macro marks the beginning of a paragraph. By default, paragraphs are blocked at the left margin and preceded by a blank line.

1.1 Second-Level Heading

Both the first- and second-level headings, as well as unnumbered headings, are set apart from the text; section numbers and heading text are emboldened.

1.1.1. Third-Level Heading This is the first paragraph under the third-level heading. The section number and heading text are italicized, and the text of this paragraph immediately follows the heading text.

2. DOCUMENTATION STYLES

The .MT macro controls the documentation style. The same input was used to produce both memorandum and released paper formats. The macro call, .MT 1, was used to produce the sample Memorandum for File; the macro call, .MT 4, was used to produce the sample released paper.

S. P. LeName

MH-12345-SPL-cl **C. Walker**

Sample Input: External Letter (.MT 5)

The following input was used to produce the sample External Letter (using **.MT 5**) shown in the next section.

```
.ND "December 7, 1981"
.TL
.nf
.bf 2
.AU "S. P. LeName" SPL MH 12345 6789 1A-100
.MT 5 "B. I. Andover"
.DS
Ms. B. I. Andover
TypeKing, Incorporated
563 Whitman Avenue
Orange, Connecticut  06477
.DE
Dear Ms. Andover:
.P
Please send me the technical information package
for product SPA-V.
.P
Since my company is considering applying for an
exclusive license to redevelop your font digitization
procedure, I would also appreciate the following
information:
.BL
.LI
What is the patent status of your procedure?
.LI
Can the license be awarded for five years?
.LI
What royalties would be required on sales?
.LE
.P
Thank you.
.FC
.SG cl
```

Sample Output: External Letter (.MT 5)

December 7, 1981

Ms. B. I. Andover
TypeKing, Incorporated
563 Whitman Avenue
Orange, Connecticut 06477

Dear Ms. Andover:

Please send me the technical information package for product SPA-V.

Since my company is considering applying for an exclusive license to redevelop your font digitization procedure, I would also appreciate the following information:

- What is the patent status of your procedure?
- Can the license be awarded for five years?
- What royalties would be required on sales?

Thank you.

Yours very truly,

SPL:cl

S. P. LeName

COVER SHEET INSTRUCTIONS

The following input can be used as a template for generating a true TM cover sheet:

.ND *date*
.TL *charging_case filing_case*
one or more lines
of title text
.AU *name initials location department extension room*
.TM *number*
.AS 1
one or more lines
of abstract text
.AE
.OK *keyword1 keyword2 keyword3 ...*
.MT
.CS *pgs_text pgs_other pgs_total #_figs #_tabs #_refs*

The input file for the cover sheet should then be processed using **troff** and the **-mm** macros; output should be sent to the phototypesetter.

NOTES

Chapter 4

VIEWGRAPH MACROS USER GUIDE

PAGE

Chapter 4

VIEWGRAPH MACROS USER GUIDE

1. Introduction

This chapter describes a package of UNIX system **troff** formatter macros called MV designed for typesetting viewgraphs and slides. It is assumed that the reader has a basic knowledge of the UNIX operating system, the text editor **ed**, and the **troff** formatter.

> ***Note 1:*** The following list contains the commands identified in this guide. In addition, the list categorizes the commands by the reference manual in which they can be found.
>
> 1. *UNIX System User Reference Manual* — **ed** and **spell**.
>
> 2. *Introduction and Reference Manual* — **eqn**, **mv**, **mvt**, **ocw**, **otroff**, **tbl**, and **troff**.
>
> ***Note 2:*** Throughout this chapter, a reference to **troff** (device independent) also means **otroff** (old troff) unless otherwise indicated.

With the MV macros, viewgraphs can be prepared in a variety of dimensions, as well as 35-mm slides and 2- by 2-inch "super-slides". These transparencies can be made in a variety of styles, in different fonts, with oversize titles, and with highlighted subordination levels. Because text from which the foils are typeset is stored on the UNIX system, the contents of a foil can be readily changed to include new data or can be incorporated into a new presentation. Text of the foils can be passed through **spell**, or preprocessed by **eqn**, **tbl**, **ocw**, etc.

> ***Note:*** The **ocw** preprocessor can only be used with **otroff**. It is not necessary with **troff** (device independent).

It is not possible to include artwork, graphics, or multicolored text in foils made with this macro package except by manual cut-and-paste methods.

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

2. Examples

Before explaining the macros in detail, the formatting process is illustrated with some examples.

2.1 Trivial Example

The following text file is given the file name of *trivial*:

```
.Sw
Six stages of a project:
.B
Wild enthusiasm
.B
Disillusionment
.B
Total confusion
.B
Search for the guilty
.B
Punishment of the innocent
.B
Promotion of the nonparticipants
```

The **.Sw** is a foil-start macro and is defined in paragraph 3.1. The following UNIX operating system command generates the viewgraph illustrated in Figure 4-1:

```
mvt trivial
```

2.2 Less Trivial Example

The foil that results from typesetting the following input is illustrated in Figure 4-2:

```
.Vw 2 " Less Trivial" " June 29, 1980"
.T " What the Walrus Said"
"The time has come," the Walrus said,
.BR
"To talk of many things:
.I .5
.B
Of shoes\(emand ships\(emand sealing wax\(em
.B
Of cabbages\(emand kings\(em
.B
And why the sea is boiling hot\(em
.B
And whether pigs have wings."
```

The **.Vw** {3.1} is another foil-start macro. Other macros (.T, .BR, and .I) in this example will be explained later. The "**\(em**" string is the **troff** formatter name for the "em dash" (long dash).

2.3 Other Examples

The inputs that produced Figures 4-3 through 4-7 are shown in this section. These foils illustrate the effect of macros that are discussed in the next section {3}.

The input for Figure 4-3 is:

```
.Vh 3 " Levels & Marks"
.T " Foil Levels & Level Marks"
This is the .A (left margin) level;
.B
this is the .B level,
.B
as is this;
.C
this is the .C level,
.C
as is this;
.D
and this is the .D level,
.D
as is this.
.A
The large bullet, the dash, and the small
bullet are the default "marks" for
levels .B, .C, and .D, respectively.
However, these three levels can also
be marked arbitrarily:
.B B.
Like this (this is the .B level);
.C 3.
like this (this is the .C level);
.D d.
like this (this is the .D level), or
.D iv.
like this, or even
\&.D \(rh~\(bu +4
like this.
.A
The .A level cannot be marked.
.B
An arbitrary number of lines of text
can be included in any item at any level;
the text will be filled, but neither adjusted
nor hyphenated, just like this .B level item.
```

The input for Figure 4-4 is:

```
.DF 1 R
.VS 4 Complex
.T " Of Bits & Bytes & Words"
.S -4
.I 3 A x
.ft I
But let your communication be, Yea, yea;
Nay, nay: for whatsoever is more than these
cometh of evil.*
.ft
.I +1 a nospace
Matthew 5:37
.BR
.S
.I 0 .A
Binary notation has been around for a
.S +6
long
.S
time.
.B
The above verse tells us to use:
.C 1)
Binary notation,
.ft I
and
.ft
.C 2)
Redundancy
.D \(rh
(in communicating)
.B
Binary notation is
.U not
suited for human use, above verse to
the contrary notwithstanding.
.SP
.S -2
.TS
box ;
c | c | c | c
```

```
l|c|c|c.
System@Bits/Byte@Bytes/Word@Bits/Word
_
IBM 7090/94@6@6@36
IBM 360/370@8@4@32
PDP 11/70@8@2@16
.TE
.S
.S -4
.U ~~~~~~~~~~~
.BR
* The use of this verse in this context
is plagiarized from C. Shannon.
.S
```

The input for Figure 4-5 is:

```
.de CW
.I .5 a
.NF
.ft 3
..
.de CN
.FI
.I 0 a
.ft 1
..
.DF 1 R 2 I 3 CW
.VS 5 " CW & EQN"
.EQ
gsize 18
.EN
.S 100 5.5
Input:
.CW
.EQ
sum from k=1 to inf m sup k-1
~=~ 1 over 1-m
.EN
.CN
```

```
Output:
.I 2 a
.EQ
sum from k=1 to inf m sup k-1
~=~ 1 over 1-m
.EN
.I 0 a
Input:
.CW
The equation $ f(t) ~=~ 2 pi
int sin ( omega t ) dt $
is used here in running text,
rather than being displayed.
.CN
Output:
.I .5 a
.EQ
delim $$
.EN
.AD
The equation $ f(t) ~=~ 2 pi
int sin ( omega t ) dt $
is used here in running text,
rather than being displayed.
.EQ
delim off
gsize 10
.EN
```

The input for Figure 4-6 is:

```
.de CW
.I .5 a
.NF
.ft 3
..
.de CN
.I 0 a
.FI
```

```
.ft 1
..
.DF 1 R 2 I 3 CW
.VS 6 " The Works: Input"
Input:
.S -4
.CW
.TS
center doublebox ;
Cip+4 ¦ Cip+4 S S
^ ¦ L L L
^ ¦ C ¦ C ¦ C
^ ¦ C ¦ C ¦ C
Li ¦ C ¦ C ¦ N .
Users Ⓣ Hardware
Ⓣ _ Ⓣ _ Ⓣ _
Ⓣ UNIX\*(Tm Ⓣ Model Ⓣ Serial
Ⓣ System Ⓣ \^ Ⓣ Number
=
OS Dev. Ⓣ A Ⓣ VAX Ⓣ 54
SGS Dev. Ⓣ B Ⓣ 11/70 Ⓣ 3275
Low-End Ⓣ C Ⓣ 11/23 Ⓣ 221
_
And now ... Ⓣ T{
.NA
Some filled text and an equation:
T} Ⓣ T{
$ zeta (s) = prod
from k=1 to inf k sup -s $
.AD
T} Ⓣ 1.2
.TE
.CN
```

The input for Figure 4-7 is:

```
.VS 7 " The Works: Output"
.EQ
delim $$
gsize 14
.EN
Output:
.I 0 a
.SP
.TS
center doublebox ;
Cip+4 ¦ Cip+4 S S
^ ¦ L L L
^ ¦ C ¦ C ¦ C
^ ¦ C ¦ C ¦ C
Li ¦ C ¦ C ¦ N .
Users Hardware
_ _ _
UNIX\*(Tm Model Serial
System \^ Number
=
OS Dev. A VAX 54
SGS Dev. B 11/70 3275
Low-End C 11/23 221
_
And now ... T{
.NA
Some filled text and an equation:
T} T{
$ zeta (s) = prod
from k=1 to inf k sup -s $
.AD
T} 1.2
.TE
.EQ
delim off
gsize 10
.EN
```

3. Macros

This section contains explanations of the MV macros which are summarized in **mv** of the *UNIX System User Reference Manual.*

3.1 Foil-Start Macros

Each foil must start with a foil-start macro. There are nine foil-start macros for generating nine different-sized foils; the names (and the corresponding mounting-frame sizes) of these macros are shown in Figure 4-8.

The naming convention for these nine macros is that the first character of the name (V or S) distinguishes between viewgraphs and slides, while the second character indicates whether the foil is square (S), small wide (w), small high (h), big wide (W), or big high (H). Slides are thinner than the corresponding viewgraphs; therefore, the ratio of the longer dimension to the shorter one is larger for slides than for viewgraphs. As a result, slide foils can be used for viewgraphs, but not vice versa. On the other hand, viewgraphs can accommodate a bit more text.

> ***Note:*** The **.VW** and **.SW** macros produce foils that are 7 by 5.4 inches because commonly available typesetter paper is less than 9 inches wide. These foils must be enlarged by a factor of 9/7 before they can be used as 9-inch wide by 7-inch high viewgraphs.

Each foil-start macro causes the previous foil (if any) to be terminated, foil separators to be produced, and certain heading information to be generated. The default heading information consists of three lines of right-justified data:

- The current date in the form *mo/dy/yr*
- BTL
- FOIL *n*.

where *n* is the sequence number in the current "run". As explained below, this heading information is replaced by the three arguments of the foil-start macro if those arguments are given.

The actual projection area is marked by "cross hairs" (plus signs) that fit into the corners of the viewgraph mount. This is an aid in positioning the foil for mounting.

All foils other than the square (**.VS**) foil also have a set of horizontal and vertical "crop marks". These indicate how much of the foil will be seen if it is made into a slide, rather than into a viewgraph.

Default heading information can be changed by specifying three optional arguments to the foil-start macro. Square brackets ([ ]) indicate that the argument they enclose is optional.

.XX [n] [id] [date]

where:

- **XX** stands for one of the nine foil-start macros
- *n* is the foil identifier (typically a number)
- *id* is other identifying information (typically the initials of the person creating the foil)
- *date* is usually the date.

The resulting heading information consists of three lines of right-justified text:

- *id*
- *date*
- FOIL *n*.

If *date* and *id* are omitted on a foil-start macro, then the corresponding values (if any) from the previous foil-start macro are used.

3.2 Level Macros

The MV macros provide four levels of indentation, called .A, .B, .C, and .D. Each of these level macros causes the text that follows it to be placed at the corresponding level of indentation.

The amount of vertical spacing done by each level macro can be changed with the .DV macro {3.7}. Figure 4.3 shows examples of the level macros.

3.2.1 The .A Level

.A [x]

The leftmost level (left margin) is obtained by the **.A** macro. The .A level is automatically invoked by each of the foil-start macros. Each .A macro spaces one half of a vertical space from the preceding text, unless the *x* argument is specified (*x* can be any character or string of characters); *x* suppresses the spacing.

The .A macro does not generate a mark of any sort; it is the "left-margin" macro. Repeated .A calls are ignored, but each successive call of any of the other three level macros generates the corresponding mark.

The .A macro can also be invoked through the .I macro {3.4}.

3.2.2 The .B Level

.B [mark [size]]

The .B level items are marked by a bullet (in slightly reduced point size). The text that follows the **.B** macro is spaced one half of a vertical space from the preceding text.

The .B level *mark* may be changed by specifying the desired character string as the first argument.

> ***Note:*** All character-string arguments that contain spaces must be quoted ("...").

Without the second argument (*size*), the point size of the *mark* is not reduced. Thus, the following will produce a numbered list:

```
.VS
This is a list of things:
.B 1.
This is thing number 1.
.B 2.
This is thing number 2.
.B 3.
This is the third and last thing on this foil.
```

It is possible to change the point size of the *mark* with the second argument (*size*). If given, it specifies the desired point-size change. An unsigned or positive (+) argument is taken as an increment; a negative (-) argument is a decrement. An argument greater than 99 causes the *mark* to be reduced in size just as if it were the default *mark*, namely, the bullet. After the *mark* is printed, the previous point size is restored. All these point-size changes are completely invisible to the user.

3.2.3 The .C Level

```
.C [ mark [ size ] ]
```

The **.C** level is like the .B level except that it is indented farther to the right and the default *mark* is a long dash (\(em) in a slightly reduced point size.

3.2.4 The .D Level

.D [mark [size]]

The **.D** level is indented farther to the right than the .C level and does not space from the previous text. It causes the text that follows to start on a new line. In other words, it causes a break {3.10}. Otherwise, it behaves like the .B and .C levels. The .D level default *mark* is a bullet smaller than that used for the .B level.

3.3 Titles

.T string

The **.T** macro creates a centered title from its argument (*string*). The argument must be enclosed within double quotes ("...") if it contains spaces. The size of the title is four points larger than the prevailing point size. Any indentation established by the .I macro {3.4} has no effect on titles; they are always centered within the foil horizontal dimension.

Figures 4-2, 4-3, and 4-4 illustrate the .T macro.

3.4 Global Indents

.I [indent] [a [x]]

The entire text (except titles) of the foil may be shifted right or left by the **.I** macro. The first argument (*indent*) is the amount of indentation that is to be used to establish a new left margin. This argument may be signed positive or negative, indicating right or left movement from the current margin. If unsigned, the argument specifies the new margin, relative to the initial default margin. If the argument is not dimensioned, it is assumed to be in inches (see *Text Formatters Reference* for legal **troff** formatter units). If the argument is null or omitted, 0i is assumed causing the margin to revert to the initial default margin.

If a second argument is specified, the .I macro calls the .A macro {3.2.1} before exiting. The third argument, if present, is passed to the .A macro.

Figures 4-2, 4-4, 4-5, 4-6, and 4-7 illustrate the .I macro.

3.5 Point Sizes and Line Lengths

.S [ps] [ll]

Each foil-start macro begins the foil with an appropriate default point size and line length. Default point sizes for each type of foil and corresponding maximum number of lines are given in Figure 4-9. Prevailing point size and line length may be changed by invoking the **.S** macro. If the *ps* argument is null, the previous point size is restored. If *ps* is signed negative, the point size is decremented by the specified amount. If *ps* is signed positive, it is used as an increment; and if *ps* is unsigned, it is used as the new point size. If *ps* is greater than 99, the initial default point size is restored (Figure 4-9). Vertical spacing is always 1.25 times the current point size.

The second argument (*ll*), if given, specifies line length. It may be dimensioned. If it is not dimensioned and is less than 10, it is taken as inches. If it is not dimensioned and is greater than or equal to 10, it is taken as **troff** formatter units {7.3}.

Figures 4-4, 4-5, and 4-6 illustrate the .S macro.

3.6 Default Fonts

.DF n font [n font ...]

The MV macros assume that the Helvetica Regular (also known as Geneva) font, mounted in position 1, is the default font. Additional fonts can be mounted and the default font can be changed. The **.DF** macro informs the **troff** formatter that *font* is in position *n*. The first-named font is the default font. Up to four pairs of arguments may be specified.

The .DF macro must immediately precede a foil-start macro; the initial setting is equivalent to

.DF 1 H 2 I 3 B 4 S

Figures 4-4, 4-5, and 4-6 illustrate the .DF macro.

3.7 Default Vertical Space

.DV [a] [b] [c] [d]

The default vertical space macro (**.DV**) allows changing the vertical spacing done by each of the four level macros {3.2}. The first argument (*a*) is the spacing for the .A macro, *b* is for the .B macro, *c* is for the .C macro, and *d* is for the .D macro. All nonnull arguments must be dimensioned. Null arguments leave the corresponding spacing unaffected. The initial setting is equivalent to

.DV .5v .5v .5v 0v

3.8 Underlining

.U string1 [string2]

The underline macro (**.U**) takes one or two arguments. The first argument (*string1*) is the string of characters to be underlined. The second argument (*string2*), if present, is not underlined but concatenated to the first argument.

For example:

.U phototypesetter

produces

phototypesetter

while

.U under line

produces

underline

Figure 4-4 illustrates the .U macro.

3.9 Synonyms

The MV macro package recognizes the .AD, .BR, .CE, .FI, .HY, .NA, .NF, .NH, .NX, .SO, .SP, .TA, and .TI uppercase text synonyms for the corresponding lowercase **troff** formatter requests. The *NROFF and TROFF User Manual* (In *Text Formatters Reference*) contains definitions of these requests.

3.10 Breaks

The .S, .DF, .DV, and .U macros do not cause a break. The .I macro causes a break only if it is invoked with more than one argument. All other MV macros always cause a break. The **troff** formatter synonyms (3.9) .AD, .BR, .CE, .FI, .NA, .NF, .SP, and .TI also cause a break.

3.11 Text Filling, Adjusting, and Hyphenation

By default, the MV macros fill, but neither adjust nor hyphenate text. This is an aesthetic judgement that seems correct for foils. These defaults can, of course, be changed by using the .AD, .FI, .HY, .NA, .NF, and .NH macros {3.9}.

4. The *troff* Preprocessors

It is possible to use the various **troff** formatter preprocessors to typeset foils that require more powerful formatting capabilities.

4.1 Tables

The **tbl** program can be used to set up columns of data within a viewgraph or slide. The .TS and .TE macros are not defined in the MV macro package, but are merely flags to **tbl**. Figures 4-4 and 4-7 illustrate the **tbl** program use.

4.2 Mathematical Expressions

The **eqn** program can be used to typeset mathematical expressions and formulas on foils provided care is taken to specify proper fonts and point sizes. The .EQ and .EN macros are not defined in the MV macro package. Figures 4-5 and 4-7 illustrate the **eqn** program.

4.3 Constant-Width Program Examples

The constant-width font simulates computer-terminal and line-printer output and can, at times, be effective in presenting computer-related topics. The **ocw** program (see manual page), as well as Figures 4-5 and 4-6 illustrate the constant-width font.

> ***Note:*** The **ocw** preprocessor is not needed with device independent **troff**.

If you are using device independent troff, your typesetter may have a constant width font available. In that case, do not use the **ocw** preprocessor. Use the **.DF** macro to define the font position.

For example:

```
.DF 1 R 2 I 3 CW
```

Then, define the .CW and .CN macros to include the font change, as shown below.

```
.de CW
.NF
.ft 3
..
.de CN
.FI
.ft 1
..
```

5. Finished Product

5.1 Phototypesetter Output

mvt [options] file ...

Typeset output is obtained via the **mvt** command. The *file* argument contains text and macro invocations for the foils. The *options* argument can be one or more of the following:

-a preview output on a terminal (other than a TEKTRONIX 4014 {5.2})

-e invoke **eqn**

-t invoke **tbl**

-T*term* direct output to *term*, where *term* can be one of the following:

st	STARE
4014	TEKTRONIX 4014
vp	Versatec printer

Using a hyphen (-) in place of *file* causes the **mvt** command to read the standard input (rather than a file), as in the following example using the **ocw** preprocessor {4.3}:

ocw [options] file ... | mvt [options] -

5.2 Output Approximation on a Terminal

mvt -a file_name ...

An approximation of the typeset output can be obtained with the **mvt** command. The resulting output shows the formatted foils except that:

- Point-size changes are not visible.
- Font changes cannot be seen.
- Titles that are too long appear proper.
- All horizontal motions are reduced to one horizontal space to the right.
- All vertical motions are reduced to one vertical space down.

For example, it appears that lines of text following a .B, .C, or .D macro do not align properly (even though, in fact, they do).

Although alignment cannot be determined from this approximation, line breaks and the amount of vertical space used by the text can be observed. If the foil is not full, the macro package prints the number of blank lines (in the then current point size) that remain on the foil; if the foil is full, a warning is printed. If the text did overflow the foil, the text will be printed after the "cross hairs."

5.3 Making Actual Viewgraphs and Slides

Output of the typesetter is so-called "mechanical paper," which is white, opaque photographic paper with black letters. There are several simple processes (for example, Thermofax, Bruning) for making transparent foils from opaque paper. Because some of these processes involve heat and because mechanical paper is heat sensitive, one should first make copies of the typesetter output on a good-quality office copier and then use these copies for making transparencies.

Getting slides made is a much more complicated photographic process that is best left to professionals. It is possible to make both positive (opaque letters on transparent background) and negative (transparent letters on opaque background) slides, as well as colored-background slides, etc.

6. Suggestions For Use

The following suggestions have been derived from experience, from the examination of several other macro packages for making foils, and from some publications that discuss good and bad foil-making practices:

- The most useful foil sizes are .VS and .Vw (or .Sw). This is because most projection screens are either square or wider than they are tall and also because the resulting foils are smaller, easier to carry, and require no enlargement before use.

- Reducing point size below the default value should be avoided. Default point size for each type of foil (Figure 4-9) is the smallest point size that will result in a foil that is legible by an audience of more than a dozen people. If there is more text than fits onto a foil, two or more foils should be used instead of reducing the point size.

- Numerous font changes should be avoided. A foil with more than two typefaces looks cluttered and distracts the viewer.

- Underlined typeset text should be avoided. Even though this package contains a macro for underlining, it should not be used. Underlined typeset text almost always looks bad; instead use a different typeface.

- The Helvetica sans-serif font is thicker and easier to read than the Times Roman serif font normally used for typesetting. On the other hand, the Times Roman font permits more text to be squeezed onto a foil. If it is intended to use italic and/or bold typefaces, either the Helvetica regular, italic, and medium* :

```
.DF 1 H 2 HI 3 HM
```

* Helvetica medium is really a bold typeface.

or the Times Roman regular, italic, and bold:

```
.DF 1 R 2 I 3 B
```

should be mounted via the .DF macro {3.6}. Bold typefaces tend to be a bit overwhelming. Choice of fonts is primarily a matter of personal choice. The following table identifies fonts used in the examples of Fig. 4-1 through 4-7.

FIGURE	*FONT*
4.1, 4.2, 4.3	H (default)
4.4, 4.7	R and I
4.5, 4.6	R and CW

- The .SP macro can be used to insert a bit of additional white space (for instance, .5v or 1v, where v means "vertical space") at the top of each foil (that is, increase the top margin).

- Normal uppercase and lowercase text is more legible than uppercase text only.* Uppercase and lowercase alphabets have evolved and have been used for many years because they result in more legible text. Furthermore, such text is less bulky than uppercase text only, so more information can be put onto a foil without crowding.

- Foils for a presentation should be made as consistent as possible. Changing fonts, typefaces, point sizes, etc., from foil to foil tends to distract the viewer. While it is possible to introduce emphasis and draw the viewer's attention to particular items with such changes, this works only if it is done purposefully and sparingly. Overuse of these techniques is almost always counter-productive.

---

* The only exceptions to this rule are foils set in a point size so small that lowercase characters simply can not be read. This is usually the case for foils produced on a normal typewriter.

In summary, the dictum that "the medium is the message" does not apply to foil making. When in doubt:

- Do not change point sizes.
- Do not change fonts or typefaces.
- Do not underline.
- Use many "sparse" foils rather than a few "dense" ones.
- Use fewer words rather than more words.
- Use larger point sizes rather than smaller point sizes.
- Use larger top and bottom margins rather than smaller ones.
- Use normal uppercase and lowercase text rather than uppercase text only.

7. Warnings

7.1 Use of *troff* Formatter Requests

In general, it is not advisable to intermix arbitrary **troff** formatter requests with the MV macros because this often leads to undesirable (and sometimes astonishing) results. The "safe" requests are ones for which uppercase text synonyms have been defined in the MV package {3.9}. Other **troff** formatter requests should be used sparingly (if at all) and with care and discipline. Be particularly careful when using requests that affect point size, indentation, page offset, line and title lengths, and vertical spacing between lines. The .I and .S macros should be used instead {3.4 and 3.5}.

7.2 Reserved Names

Certain names are used internally by this macro package. In particular, all 2-character names starting with either ")" or "]" are reserved. Names that are the same as names of the MV macros and strings described in this part or names that are the same as **troff** names cannot be used. Furthermore, if any of the preprocessors {4.1, 4.2, and 4.3} are used, their reserved names must also be avoided.

7.3 Miscellaneous

The .S macro changes the point size and vertical spacing immediately, but a line-length change requested with that macro does not take effect until the next-level macro call. Specifying a third argument to the .S macro usually results in a disaster.

The "*(Tm" string generates a trademark symbol.

The tilde (~) is defined by the MV macros as a "nonpaddable" space; that is, the tilde may be used wherever a fixed-size (non adjustable) space is desired. To override this condition, the following line should be included in the input file:

```
.tr ~~
```

8. Dimensional Details

For each style of viewgraph, Figure 4-9 shows the default point size; the maximum number of lines of text (at the default point size); and the height, width, and aspect ratio, both nominal and actual.

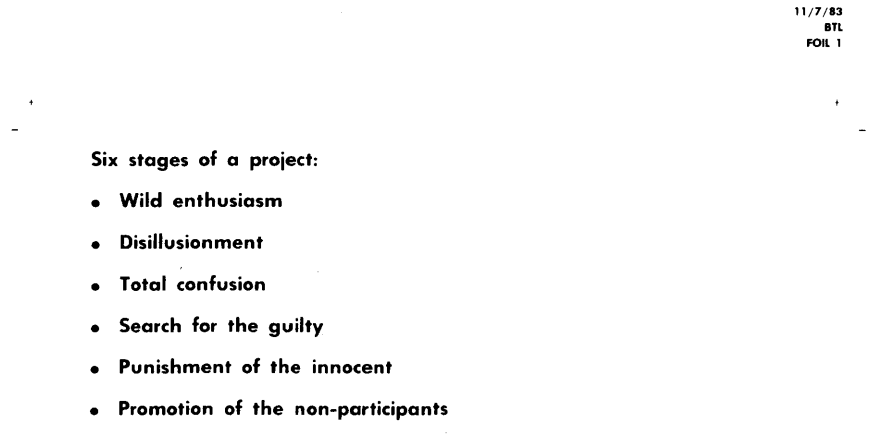


Figure 4-1. Trivial Example (Not Actual Size)

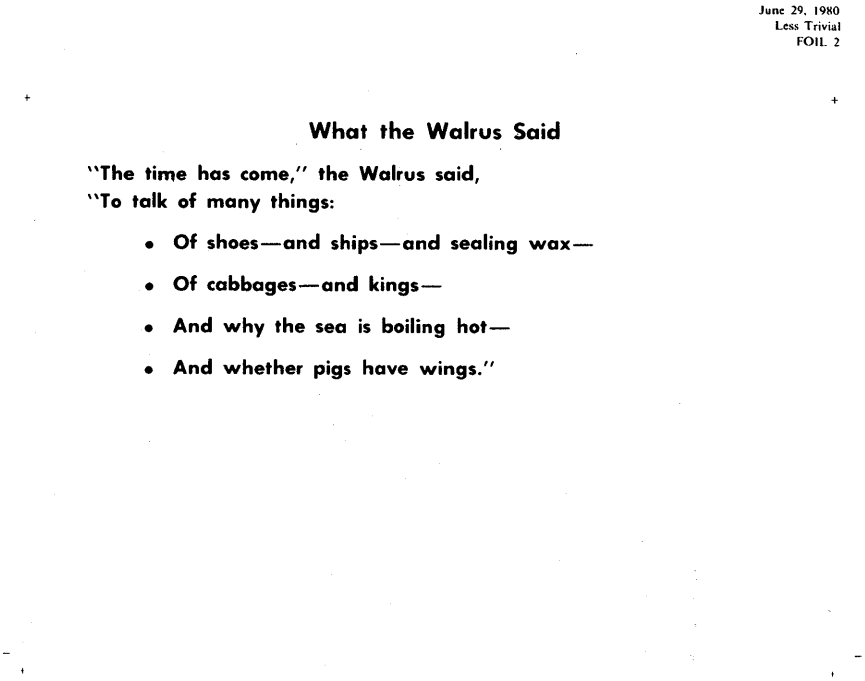


Figure 4-2. Less Trivial Example (Not Actual Size)

11/4/83
Levels & Marks
FOIL 3

Foil Levels & Level Marks

This is the .A (left margin) level;

- this is the .B level,
- as is this;
  - — this is the .C level,
  - — as is this;
    - and this is the .D level,
    - as is this.

The large bullet, the dash, and the small bullet are the default "marks" for levels .B, .C, and .D, respectively. However, these three levels can also be marked arbitrarily:

B. Like this (this is the .B level);

3. like this (this is the .C level);
   d. like this (this is the .D level), or
   iv. like this, or even
   ● like this.

The .A level cannot be marked.

- An arbitrary number of lines of text can be included in any item at any level; the text will be filled, but neither adjusted nor hyphenated, just like this .B level item.

Figure 4-3. Example of Foil Levels (Not Actual Size)

11/7/83
Complex
FOIL 4

Of Bits & Bytes & Words

*But let your communication be, Yea, yea; Nay, nay: for whatsoever is more than these cometh of evil.**
Matthew 5:37

Binary notation has been around for a long time.

- The above verse tells us to use:

  1) Binary notation, *and*

  2) Redundancy
  (in communicating)

- Binary notation is not suited for human use, above verse to the contrary notwithstanding.

System	Bits/Byte	Bytes/Word	Bits/Word
IBM 7090/94	6	6	36
IBM 360/370	8	4	32
PDP 11/70	8	2	16

* The use of this verse in this context is plagiarized from C. Shannon.

Figure 4-4. Example of a Square Foil (Not Actual Size)

11/4/83
CW & EQN
FOIL 5

Input:

```
.EQ
sum from k=1 to inf m sup k-1
  =   1 over 1-m
.EN
```

Output:

$$\sum_{k=1}^{\infty} m^{k-1} = \frac{1}{1-m}$$

Input:

```
The equation $ f(t)  =  2 pi
int sin ( omega t ) dt $
is used here in running text,
rather than being displayed.
```

Output:

The equation $f(t) = 2\pi\int\sin(\omega t)dt$ is used here in running text, rather than being displayed.

Figure 4-5. Example of Indent (Not Actual Size)

11/4/83
The Works: Input
FOIL 6

Input:

```
.TS          (Ⓣ  = tab)
center doublebox ;
Cip+4  | Cip+4  S  S
^  | L  L  L
^  | C  | C  | C
^  | C  | C  | C
Li  | C  | C  | N
Users Ⓣ Hardware
Ⓣ _Ⓣ _Ⓣ _
Ⓣ UNIX\*(Tm Ⓣ Model Ⓣ Serial
Ⓣ System Ⓣ  Ⓣ Number
=
OS  Dev. Ⓣ A Ⓣ VAX Ⓣ 54
SGS  Dev. Ⓣ B Ⓣ 11/70 Ⓣ 3275
Low-End Ⓣ C Ⓣ 11/23 Ⓣ 221

And now . . . Ⓣ T}
Some filled text and an equation:
T}Ⓣ T}
$ zeta (s) = prod
from k=1 to inf k sup -s $
T}Ⓣ 1.2
.TE
```

Figure 4-6. Example of Input of a Table Foil (Not Actual Size)

delim $$ gsize 14 Output:

Users	*Hardware*		
	UNIX™ System	Model	Serial Number
OS Dev.	A	VAX	54
SGS Dev.	B	11/70	3275
Low-End	C	11/23	221
And now ...	Some filled text and an equation:	$ zeta (s) = prod from k=1 to inf k sup -s $	1.2

delim off gsize 10

Figure 4-7. Example of Output of a Table Foil (Not Actual Size)

MACRO NAME	*SIZE* AND TYPE*
.VS	7X7 viewgraph or 2X2 super-slide
.Vw	7X5 viewgraph
.Vh	5X7 viewgraph
.VW	9X7 viewgraph
.VH	7X9 viewgraph
.Sw	7X5 35-mm slide
.Sh	5X7 35-mm slide
.SW	9X7 35-mm slide
.SH	7X9 35-mm slide

* Size of mounting frame opening (width and height) in inches.

Figure 4-8. Table of Foil-Start Macros

MACRO (NOTE 1)	*POINT SIZE*	*MAXIMUM LINES (NOTE 2)*	*NOMINAL*				*ACTUAL (TEXT)*			
			W	*H*	*AR*	1/AR	*W*	*H*	*AR*	1/AR
			— (NOTE 3) —				*— (NOTE 3) —*			
.VS	18	21	7	7	1	1	6	6.8	1.13	.88
.Vw	14	19	7	5	.71	1.4	6	4.8	.8	1.25
.Vh	14	27	5	7	1.4	.71	4.2	6.8	1.6	.62
.VW	14	21	7	5.4	.77	1.3	6	5.2	.87	1.15
.VH	18	28	7	9	1.3	.77	6	8.8	1.5	.68
.Sw	14	18	7	4.6	.67	1.5	6	4.4	.73	1.4
.Sh	14	27	4.6	7	1.5	.67	3.8	6.8	1.8	.56
.SW	14	18	7	4.6	.67	1.5	6	4.4	.73	1.4
.SH	18	28	6	9	1.5	.67	5	8.8	1.76	.57

Note 1: If used as a viewgraph, the .SW macro and .VW macro generated foils must be enlarged by a factor of 9/7.

Note 2: Maximum number of lines of text at the default point size.

Note 3: W—Width in inches, H—Height in inches, AR—Aspect ratio (H/W).

Figure 4-9. Default Point Size, Dimensions, and Aspect Ratios

NOTES

CONTENTS

Chapter 1

INTRODUCTION

PAGE

Chapter 1

INTRODUCTION

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

- *Introduction and Reference Manual*—Select Code 307-150
- *Text Formatters Reference*—Select Code 307-151
- *Macro Packages Reference*—Select Code 307-152

Each of the chapters in this book is a user guide to a specific text preprocessor. Information is provided in each chapter that will allow the user to understand and use the preprocessors. Numerous examples are included that will provide the user a base to build on when learning to use the preprocessors. The beginning user should refer to the DOCUMENTER'S WORKBENCH software *Introduction and Reference Manual* for a better overall description of the text processing tools available on the UNIX system.

1. Using the Preprocessors

A preprocessor allows a user to produce complicated formatted output such as tables and pictures from an input language that is easier to use than the formatter language. For example, a complicated, multi-column table that is boxed on all sides, with each item properly aligned and boxed, can be produced with only a few lines of input text and the table data using the **tbl** preprocessor. To produce the same output using only the formatter requests would be much more difficult and time consuming.

---

* UNIX is a trademark of AT&T Bell Laboratories.

† DOCUMENTER'S WORKBENCH is a trademark of AT&T Technologies.

To use the preprocessors, the unformatted text file is written using macros or formatter requests with the input destined for the preprocessor set off by delimiting macros or characters. The preprocessor works on the unformatted file first, replacing the text between the delimiters with the text formatter requests that produce the desired output. The output from the preprocessor is then processed by a text formatter. Normally, this is done using the piping mechanism of the UNIX system. For example:

```
tbl file | eqn | troff
```

would be the command line used to format with troff a file containing tables and equations. Note that several preprocessors can be used in the same process because each has its own language and each one only expands input found between its own delimiters.

2. Preprocessors Covered

The following preprocessors are covered in this book.

- Chapter 2—*Table Formatting Program* (tbl)
- Chapter 3—*Picture Graphics Language* (pic)
- Chapter 4—*Mathematics Typesetting Program* (eqn)

Chapter 2

TABLE FORMATTING PROGRAM

PAGE

Chapter 2

TABLE FORMATTING PROGRAM

1. Introduction

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

A description of a table is translated by the **tbl** program into a list of **nroff/troff** formatter requests that will produce the table. The **tbl** program isolates a portion of a job that it can successfully handle (text between the **.TS** and **.TE** delimiting macros) and leaves the remainder for other programs. Thus, **tbl** may be used with the equation formatting program (**eqn**), the graphics formatting program (**pic**), and/or various formatter layout macro packages without function duplication.

2. Usage

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

tbl *filename*|troff

When there are several input files containing tables, equations, pictures, and *mm* macro requests, the normal command is

tbl file1 file2...|eqn|troff -mm

The usual options may be used on the **troff** formatter. Usage of the **nroff** formatter is similar to that of **troff**. If a file name is "−",

the standard input is read at that point.

For the convenience of users employing line printers without adequate driving tables or post-filters, there is a special *-TX* command-line option to **tbl** which produces output that does not have fractional line motions.

When both **tbl** and **eqn** programs operate on the same file, **tbl** should be called first. If there are no equations within tables, either sequence works. It is usually faster to execute **tbl** first since **eqn** normally produces a larger expansion of the input. However, if there are equations within tables (using the *delim* statement in **eqn**), **tbl** must be executed first or the output will be scrambled. Use of equations in **n**-style (numeric) columns should be avoided since **tbl** attempts to split numerical format items into two parts. The *delim (xy)* global option prevents splitting numerical columns within delimiters. For example, if the **eqn** delimiters are "$$", a *delim (*$$*)* statement causes a numerical column such as

1245 $\pm$ 16$

to be divided after 1245, not after 16.

The **tbl** program accepts up to 35 columns; the actual number that can be processed may be smaller depending on availability of **troff** formatter number registers. Number register names used by **tbl** must be avoided within tables. These include 2-digit numbers from 31 to 99 and strings of the form 4*x*, 5*x*, #*x*, *x*+, *x*|, ^*x*, and *x*–, where *x* is any lowercase letter. The names ##, #–, and #^ are also used in certain circumstances. To conserve register names, the **n** and **a** key letters (key letters are introduced in paragraph 3.2) share a register. Hence, the restriction that they may not be used in the same column.

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

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

```
.TS H
```

then, a line of the form

```
.TH
```

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

3. Input Commands

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

The general format of the input is

text
.TS
table
.TE
text
.TS
table
.TE
text

The format of each table is

.TS
options;
format.
data
.TE

Each table is independent and contains:

- Global options {3.1}
- A format section describing individual columns and rows of the table {3.2}
- Data to be printed {3.3}.

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

3.1 Global Options

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

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

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

3.2 Format Section

The format section of the table specifies the layout of the columns. Each line in the format section corresponds to one line of table data (except the last format line corresponds to all following data lines up to any additional **.T&** command line). Each format line contains a **key letter** for each column of the table. Key letters may be separated by spaces or tabs for readibility purposes. Key letters are:

L or **l** — Indicates a left-adjusted column entry.

R or **r** — Indicates a right-adjusted column entry.

C or **c** — Indicates a centered column entry.

N or **n** — Indicates a numerical column entry. Numerical entries are aligned so that the units digits of numbers line up.

A or **a** — Indicates an alphabetic subcolumn. All corresponding entries are aligned on the left and positioned so that the widest entry is centered within the column.

S or **s** — Indicates a spanned heading. The entry from the previous column continues across this column (not allowed for the first column of the table).

^ — Indicates a vertically spanned heading. The entry from the previous row continues down through this row (not allowed for the first row of the table).

When numerical column alignment (**n**) is specified, a location for the decimal point is sought. The rightmost dot (**.**) adjacent to a digit is used as a decimal point. If there is no dot adjoining a digit, the rightmost digit is used as a units digit. If no alignment is indicated, the item is centered in the column. However, the special nonprinting character string **\&** may be used to override dots and digits or to align alphabetic data. This aligns the dots and the **\&** disappears from the final output.

In the following example, items shown in the **INPUT** column will be aligned (in a numerical column) as shown in the **OUTPUT** column.

INPUT:	*OUTPUT:*
.TS	
center;	
n.	
13	13
4.2	4.2
26.4.12	26.4.12
abcdefg	abcdefg
abcd\&efg	abcdefg
abcdefg\&	abcdefg
43\&3.22	433.22
749.12	749.12
.TE	

If numerical data are used in the same column with wider **L** (the capital **L** key letter is used instead of lowercase for readability) or **r** type table entries, the widest number is centered relative to the wider **L** or **r** items. Alignment within the numerical items is preserved. This is similar to the behavior of **a** type data. Alphabetic subcolumns (requested by the **a** key letter) are always slightly indented relative to **L** items. If necessary, the column width is increased to force this. This is not true for **n** type entries.

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

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

```
css
lnn.
```

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

A sample table in this format is:

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

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

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

Additional features of the key letter system are:

- ***Horizontal lines*** - A key letter may be replaced by underscore (_) to indicate a horizontal line in place of the column entry or equal (=) to indicate a double horizontal line. If an adjacent column contains a horizontal line or if there are vertical lines adjoining this column, the horizontal line is extended to meet nearby lines. If any data entry is provided for this column, it is ignored and a warning message is printed.

- ***Vertical lines*** - A vertical bar (|) placed between column key letters will cause a vertical line between the corresponding columns of the table. A vertical bar to the left of the first key letter or to the right of the last one produces a line at the edge of the table. If two vertical bars appear between key letters, a double vertical line is drawn.

- ***Space between columns*** - A number may follow the key letter indicating the amount of separation between this column and the next column. The number specifies the separation in *ens*. One *en* is about the width of the letter "n". More precisely, an *en* is the number of points (1 point = 1/72 inch) equal to half the current type size. If the *expand* option is used, these numbers are multiplied by a constant such that the table is as wide as the current line length. The default column separation number is 3. If the separation is changed, the worst case (largest space

requested) governs.

- ***Vertical spanning*** - Vertically spanned items extending over several rows of the table are centered in their vertical range. If a key letter is followed by **t** or **T**, any corresponding vertically spanned item will begin at the top line of its range.

- ***Font changes*** - A key letter followed by a string containing a font name or number preceded by the letter **f** or **F** indicates that the corresponding column should be in a different font from the default font (usually Roman). All font names are one or two letters. A 1-letter font name should be separated from whatever follows by a space or tab. The single letters **B**, **b**, **I**, and **i** are shorter synonyms for **fB** and **fI**. Font-change requests given with the table entries override these specifications.

- ***Point size changes*** - A key letter followed by **p** or **P** and a number indicates the point size of the corresponding table entries. If the number is a signed digit, it is taken as an increment or decrement from the current point size. If both a point size and a column separation value are given, one or more blanks must separate them.

- ***Vertical spacing changes*** - A key letter followed by **v** or **V** and a number indicates the vertical line spacing used within a multiline table entry. The number may be a signed digit, in which case it is taken as an increment or decrement from the current vertical spacing. A column separation value must be separated by blanks or some other specification from a vertical spacing request. This request has no effect unless the corresponding table entry is a text block.

- ***Column width indication*** - A key letter followed by **w** or **W** and a width value in parentheses indicates minimum column width. If the largest element in the column is not as wide as the width value given after the **w**, the largest element is assumed to be that wide. If the largest element in the column is wider than the specified value, its width is used. The width is also used as a default line length for included text blocks. Normal **troff** formatter units can be used to scale the width value. The default value is *ens* if none are used. If the width specification is a unitless integer, the parentheses may be omitted. If another width value is given in a column, the last one controls the width.

- ***Equal-width columns*** - A key letter followed by **e** or **E** indicates equal-width columns. All columns whose key letters are followed by **e** or **E** are made the same width. This permits a group of regularly spaced columns.

- ***Staggered columns*** - A key letter followed by **u** or **U** indicates that the corresponding entry is to be moved up one-half line. This makes it easy to have a column of differences between numbers in an adjoining column. The *allbox* option does not work with staggered columns.

- ***Zero-width item*** - A key letter followed by **z** or **Z** indicates that the corresponding data item is to be ignored in calculating column widths. This may be useful in allowing headings to run across adjacent columns where spanned headings would be inappropriate.

- ***Default*** - Column descriptors missing from the end of a format line are assumed to be **L**. The longest line in the format section, however, defines the number of columns in the table. Extra columns in the data are ignored.

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

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

3.3 Data To Be Printed

Data for the table are input after the format section. Each table line is typed as one line of data. Very long input lines can be broken. Any line whose last character is a backslash (\) is combined with the following line; i.e., the backslash vanishes. Data for different columns (table entries) are separated by tabs or by whatever character has been specified in the **tab** global option {3.1}.

There are a few special cases of data entries:

- **troff** *commands within tables* - An input line beginning with a dot and followed by anything but a number (*.xx*) is assumed to be a request to the formatter and is passed through unchanged retaining its position in the table. For example, a space within a table may be produced with the **.sp** request in the data.

- *Full width horizontal lines* - An input line containing only the _ (underscore) character or = (equal sign) is taken to be a single or double line, respectively, extending the full width of the table.

- *Single column horizontal lines* - An input table entry containing only the _ character or the = is taken to be a single or double line extending the full width of the column. Such lines are extended to meet horizontal or vertical lines adjoining this column. To obtain these characters explicitly in a column, they should be preceded by a \& or followed by a space before the usual tab or newline character.

- *Short horizontal lines* - An input table entry containing only the string _ is assumed to be a single line as wide as the contents of the column. It is not extended to meet adjoining lines.

- *Repeated characters* - An input table entry containing only a string of the form \R*x*, where *x* is any character, is replaced by repetitions of the character *x* as wide as data in the column. The sequence is not extended to meet adjoining columns.

- *Vertically spanned items* - An input table entry containing only the \^ character string indicates that the table entry immediately above spans downward over this row. It is equivalent to a table format key letter of ^.

- *Text blocks* - In order to include a block of text as a table entry, precede it by **T{** and follow it by **T}**. Thus, the sequence

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

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

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

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

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

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

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

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

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

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

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

When calculating column widths, all table entries are assumed to be in the font and size being used when the **.TS** command was

encountered. This is true except for font and size changes indicated in the table format section or within the table data (as in the entry **\s+3Data\s0**). Because arbitrary **troff** requests may be sprinkled in a table, care must be taken to avoid confusing width calculations. It is not possible to change the number of columns, the space between columns, the global options such as *box*, or the selection of columns to be made equal in width.

4. Additional Command Lines

To change the format of a table after many similar lines, as with subheadings or summarizations, the **.T&** (table continue) command is used to change column parameters. It is not recognized after the first 200 lines of a table. The outline of such a table input is

```
.TS
options;
format.
data
...
.T&
format.
data
.T&
format.
data
.TE
```

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

5. Examples

Figures 2-1 through 2-6 are included to show input and output information that illustrate the basic concepts of the **tbl** program. The ⓣ symbol in the input data represents a tab character. Although each figure has a title that indicates an option or feature, other examples of use may be gleaned from them. For instance, Figure 2-5 also indicates the requesting of bold type print in the format area.

INPUT:

```
.TS
box;
c c c
l l l.
Language⓽Authors⓽Runs on
.sp
_
Fortran⓽Many⓽Almost anything
C⓽BTL⓽11/45,H6000,370
BLISS⓽Carnegie-Mellon⓽PDP-10,11
IDS⓽Honeywell⓽H6000
Pascal⓽Stanford⓽370
.TE
```

OUTPUT:

Language	Authors	Runs on
Fortran	Many	Almost anything
C	AT&T BL	11/45,H6000,370
BLISS	Carnegie-Mellon	PDP-10,11
IDS	Honeywell	H6000
Pascal	Stanford	370

Figure 2-1. Table Using "box" Option

INPUT:

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

OUTPUT:

AT&T 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 2-2. Table Using "allbox" Option

INPUT:

```
.TS
box;
c s s
c | c | c
l | l | n.
Major New York Bridges
_
Bridge⓪Designer⓪Length
_
Brooklyn⓪J. A. Roebling⓪1595
Manhattan⓪G. Lindenthal⓪1470
Williamsburg⓪L. L. Buck⓪1600
_
Queensborough⓪Palmer &⓪1182
⓪ Hornbostel
_
⓪⓪1380
Triborough⓪O. H. Ammann⓪_
⓪⓪383
_
Bronx Whitestone⓪O. H. Ammann⓪2300
Throgs Neck⓪O. H. Ammann⓪1800
.TE
```

OUTPUT:

Major New York Bridges		
Bridge	Designer	Length
Brooklyn	J. A. Roebling	1595
Manhattan	G. Lindenthal	1470
Williamsburg	L. L. Buck	1600
Queensborough	Palmer & Hornbostel	1182
Triborough	O. H. Ammann	1380
		383
Bronx Whitestone	O. H. Ammann	2300
Throgs Neck	O. H. Ammann	1800

Figure 2-3. Table Using "vertical bar" Key Letter Feature

INPUT:

```
.TS
box;
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 2-4. Table Using Horizontal Lines In Place Of Key Letters

INPUT:

```
.TS
box;
cfB s s s.
Composition of Foods
_
.T&
c | c s s
c | c s s
c | c | c | c.
Food⒯Percent by Weight
\^⒯_
\^⒯Protein⒯Fat⒯Carbo-
\^⒯\^⒯\^⒯hydrate
_
.T&
l | n | n | n.
Apples⒯.4⒯.5⒯13.0
Halibut⒯18.4⒯5.2⒯...
Lima beans⒯7.5⒯.8⒯22.0
Milk⒯3.3⒯4.0⒯5.0
Mushrooms⒯3.5⒯.4⒯6.0
Rye bread⒯9.0⒯.6⒯52.7
.TE
```

OUTPUT:

Composition of Foods			
Food	Percent by Weight		
	Protein	Fat	Carbo-hydrate
Apples	.4	.5	13.0
Halibut	18.4	5.2	...
Lima beans	7.5	.8	22.0
Milk	3.3	4.0	5.0
Mushrooms	3.5	.4	6.0
Rye bread	9.0	.6	52.7

Figure 2-5. Table Using Additional Command Lines

INPUT:

```
.TS
allbox;
cfI s s
cw(1i) cw(1.75i) cw(1.75i)
l l l.
New York Area Rocks
.sp
Era⊕Formation⊕Age (years)
Precambrian⊕Reading Prong⊕>1 billion
Paleozoic⊕Manhattan Prong⊕400 million
Mesozoic⊕T{
.na
Newark Basin, incl.
Stockton,Lockatong, and Brunswick
formations
.ad
T}⊕200 million
Cenozoic⊕Coastal Plain⊕T{
.na
On Long Island 30,000 years;
Cretaceous sediments redeposited
by recent glaciation
.ad
T}
.TE
```

OUTPUT:

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

Figure 2-6. Table Using Text Blocks

Chapter 3

PIC GRAPHICS LANGUAGE

PAGE

Chapter 3

PIC GRAPHICS LANGUAGE

1. Introduction

Pic is a language for drawing simple pictures. It operates as yet another **troff** preprocessor, (in the same style as **eqn** and **tbl**), with pictures marked by `.PS` and `.PE`. **Pic** is a procedural language—a picture is drawn by specifying the motions that one goes through to draw it.

This document is primarily a user's manual for **pic**. Part 2 shows how to use **pic** in the most simple way. Subsequent parts describe how to change the sizes of objects when the defaults are inappropriate, and how to change their positions when the standard positioning rules are inappropriate. Part 3, *Reference Manual*, describes the **pic** language precisely.

1.1 TROFF Interface

Pic is usually run as a **troff** preprocessor using a command line as shown below:

pic [*options*] *file* ¦ troff

Run it before **eqn** and **tbl** if they are also present.

The command line options to **pic** are:

−T*xxx* Output is being prepared for the device *xxx*. The default is *xxx*=**aps** for the AUTOLOGIC, Incorporated APS-5 phototypesetter. This is the only phototypesetter currently supported by device-independent **troff**. Supported laser printers are the

IMAGEN* 10 (xxx=**i10**) and the Xerox 9700 (xxx=**x97**). Any unrecognized value is taken as the resolution in units per inch of the output device.

−D Draw all lines using the "\D" escape sequence of the device-independent **troff** formatter. This can be used to correct problems with characters that do not align properly when used to draw lines.

−d Sets a debug mode where some useful information is output with the picture code.

If the `.PS` line looks like

.PS <*file*

then the contents of *file* are inserted in place of the `.PS` line (whether or not the file contains `.PS` or `.PE`).

Other than this file inclusion facility, **pic** copies the `.PS` and `.PE` lines from input to output intact, except that it adds two things right on the same line as the `.PS`:

```
.PS h w
```

Arguments `h` and `w` are the picture height and width in units.

If "`.PF`" is used instead of `.PE`, the position after printing is restored to where it was before the picture started, instead of being at the bottom. ("`F`" is for "flyback.")

Any input line that begins with a period is assumed to be a **troff** command that makes sense at that point; it is copied to the output at that point in the document. Requests for spaces or changing the line spacing is not recommended here. They may confuse the **pic**

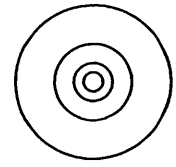
---

* IMAGEN is a registered trademark of the IMAGEN Corporation.

preprocessor. Point size and font changes are acceptable. So, for example,

```
.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    \" don't forget to restore size
```

gives



Point sizes, fonts, and local motions can be modified within quoted strings ("...") in **pic**, so long as whatever changes are made are unmade before exiting the string. For example, to print text in italic, in size 8, use

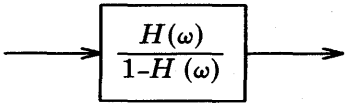
```
ellipse "\s8\fISmile!\fP\s0"
```

This produces

This is essentially the same rule as applies in **eqn**.

There is a subtle problem with complicated equations inside **pic** pictures—they come out wrong if **eqn** has to leave extra vertical space for the equation. If your equation involves more than subscripts and superscripts, you must add to the beginning of each equation the extra information `space 0`:

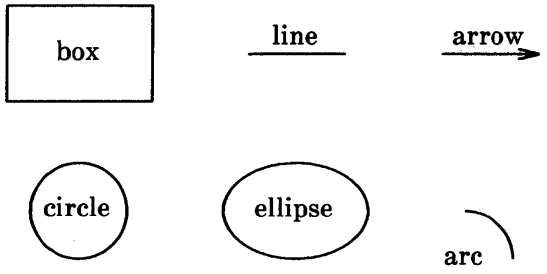
```
arrow
box "$space 0 {H( omega )} over {1 - H( omega )}$"
arrow
```



2. PIC User Manual

2.1 Basics

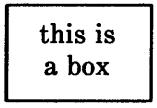
Pic provides boxes, lines, arrows, circles, ellipses, arcs, and splines (arbitrary smooth curves), plus facilities for positioning and labeling them. The picture below shows all of the fundamental objects (except for splines) in their default sizes:



Each picture begins with `.PS` and ends with `.PE`; between them are commands to describe the picture. Each command is typed on a line by itself. For example

```
.PS
box "this is" "a box"
.PE
```

creates a standard box (¾ inch wide, ½ inch high) and centers the two pieces of text in it:



Each line of text is a separate quoted string. Quotes are mandatory, even if the text contains no blanks. (Of course there needn't be any text at all.) Each line will be printed in the current size and font, centered horizontally, and separated vertically by the current **troff** line spacing. **Pic** does not center the drawing itself.

The definitions of the `.PS` and `.PE` macros for centering pictures would be:

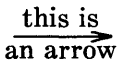
```
.de PS
.if t .sp .3
.in (\\n(.lu-\\$2u)/2u
.ne \\$1u
..
.de PE
.in
.if t .sp .6
..
```

You can use `circle` or `ellipse` in place of `box`:

Text is centered on lines and arrows; if there is more than one line of text, the lines are centered above and below:

```
.PS
arrow "this is" "an arrow"
.PE
```

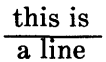
produces



and

```
line "this is" "a line"
```

gives



Boxes and lines may be dashed or dotted; just add the word `dashed` or `dotted` after `box` or `line`.

Arcs by default turn 90 degrees counterclockwise from the current direction; you can make them turn clockwise by saying `arc cw`. So

```
line; arc; arc cw; arrow
```

produces

A spline might well do this job better; we will return to that shortly.

As you might guess,

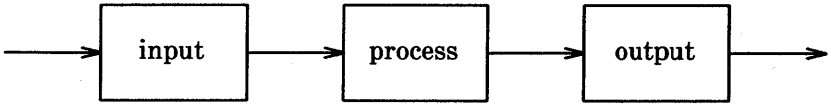
```
arc; arc; arc; arc
```

draws a circle, though not very efficiently.

Objects are normally drawn one after another, left to right, and connected at the obvious places. Thus the input

```
arrow; box "input"; arrow; box "process"; arrow; box "output"; arrow
```

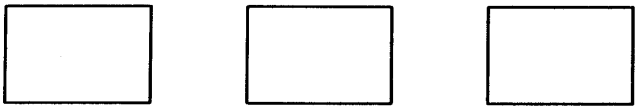
produces the figure



If you want to leave a space at some place, use `move`:

```
box; move; box; move; box
```

produces

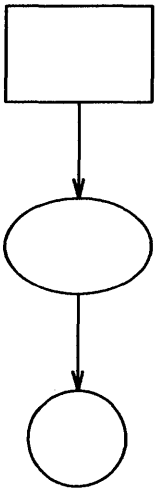


Notice that several commands can be put on a single line if they are separated by semicolons.

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; box; arrow; ellipse; arrow; circle
```

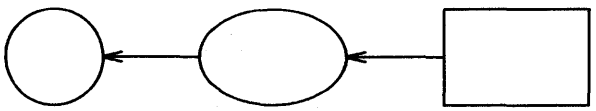
produces



and

```
left; box; arrow; ellipse; arrow; circle
```

produces



Each new picture begins going to the right.

Normally, figures are drawn at a fixed scale, with objects of a standard size. It is possible, however, to arrange that a figure be expanded 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. Thus

```
.PS 3.5i
```

causes the picture to be 3.5 inches wide.

Pic cannot produce output when the size of text is specified in relation to the size of boxes, circles, and so on. There is as yet no way to say "make a box that just fits around this text" or "make this text fit inside this circle" or "draw a line as long as this text." Tight fitting of text can generally only be done by trial and error.

If you make a grammatical error in the way you describe a picture, **pic** will complain and try to indicate where. 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 was first noted; it typically *follows* the word in error.

2.2 Controlling Sizes

This section deals with how to control the sizes of objects when the "default" sizes are not what is wanted. The next section deals with positioning them when the default positions are not right.

Each object that **pic** knows about (boxes, circles, etc.) has associated dimensions, like height, width, radius, and so on. By default, **pic** tries

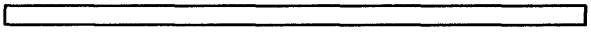
to choose sensible default values for these dimensions, so that simple pictures can be drawn with a minimum of fuss and bother. All of the figures and motions shown so far have been in their default sizes.

box	¾″ wide × ½″ high
circle	½″ diameter
ellipse	¾″ wide × ½″ high
arc	½″ radius
line or arrow	½″ long
move	½″ in the current direction

When necessary, you can make any object any size you want. For example, the input

```
box width 3i height 0.1i
```

draws a long, flat box



3 inches wide and 1/10 inch high. There must be no space between the number and the "`i`" that indicates a measurement in inches. In fact, the "`i`" is optional; all positions and dimensions are taken to be in inches.

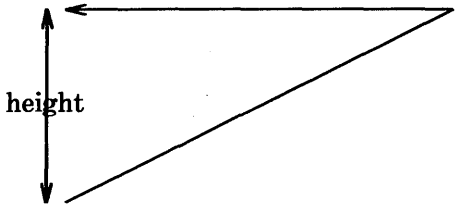
Giving an attribute like **`width`** changes only the one instance of the object. You can also change the default size for all objects of a particular type, as discussed later.

The attributes of **`height`** (which you can abbreviate to **`ht`**) and **`width`** (or **`wid`**) apply to boxes, circles, ellipses, and to the head on an arrow. The attributes of **`radius`** (or **`rad`**) and **`diameter`** (or **`diam`**) can be used for circles and arcs if they seem more natural.

Lines and arrows are most easily drawn by specifying the amount of motion from where one is right now, in terms of directions. 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 <-> down 1i "height"
.PE
```

draws

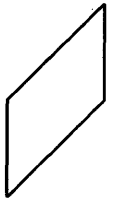


The notation <-> indicates a two-headed arrow; use -> for a head on the end and <- for one on the start. Lines and arrows are really the same thing; in fact, `arrow` is a synonym for `line ->`.

If you don't put any distance after **up**, **down**, etc., **pic** uses the standard distance. So

```
line up right; line down; line down left; line up
```

draws the parallelogram



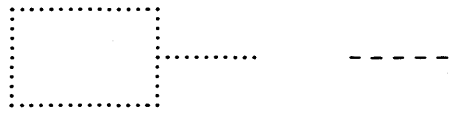
Warning: a very common error is to say

```
line 3i
```

A direction is needed.

```
line right 3i
```

Boxes and lines may be dotted or dashed:



comes from

```
box dotted; line dotted; move; line dashed
```

If there is a number after `dot`, the dots will be that far apart. You can also control the size of the dashes (at least somewhat): 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 the inputs (as separate pictures)

```
line right 4.5i dashed
line right 4.5i dashed 0.25i
line right 4.5i dashed 0.5i
line right 4.5i dashed 1i
```

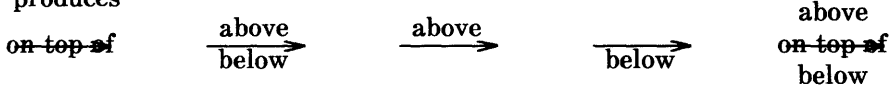
Circles and arcs cannot be dotted or dashed.

You can make any object invisible by adding the word `invis(ible)` after it. This is particularly useful for positioning things correctly near text, as we will see later.

Text may be positioned on lines and arrows:

```
.PS
arrow "on top of"; move
arrow "above" "below"; move
arrow "above" above; move
arrow "below" below; move
arrow "above" "on top of" "below"
.PE
```

produces



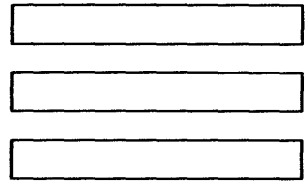
The "width" of an arrowhead is the distance across its tail; the "height" is the distance along the shaft. The arrowheads in this picture are default size.

As we said earlier, arcs go 90 degrees counterclockwise from where you are right now, and `arc cw` changes this to clockwise. The default radius is the same as for circles, but you can change it with the `rad` attribute. It is also easy to draw arcs between specific places; this will be described in the next section.

To put an arrowhead on an arc, use one of `<-`, `->` or `<->`.

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, add the word `same`. Thus in the set of boxes given by

```
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.

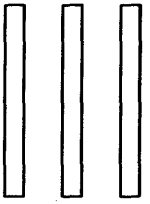
It is possible to change the default sizes of objects by assigning values to certain variables:

```
boxwid, boxht
linewid, lineht
dashwid
circlerad
arcrad
ellipsewid, ellipseht
movewid, moveht
arrowwid, arrowht     (These refer to the arrowhead.)
```

So if you want all your boxes to be long and skinny, and relatively close together,

```
boxwid = 0.1i; boxht = 1i
movewid = 0.2i
box; move; box; move; box
```

gives



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. Thus, for example, `scale=2.54` causes dimensions to be interpreted as centimeters.

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.

2.3 Controlling Positions

You can place things anywhere you want; **pic** provides a variety of ways to talk about places. **pic** uses a standard Cartesian coordinate system, so any point or object has an *x* and *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 like `from`, `to`, `by` and `at` are followed by an *x,y* pair, and can be attached to boxes, circles, lines, motions, and so on, to specify or modify a position.

You can also use `up`, `down`, `right`, and `left` with `line` and `move`. Thus

```
.PS 2
box ht 0.2 wid 0.2 at 0,0 "1"
move to 0.5,0        # or "move right 0.5"
box "2" same         # use same dimensions as last box
move same            # use same motion as before
box "3" same
.PE
```

draws three boxes, like this:

Note the use of `same` to repeat the previous dimensions instead of reverting to the default values.

Comments can be used in pictures; they begin with a `#` and end at the end of the line.

Attributes like `ht` and `wid` and positions like `at` can be written out in any order. So

```
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
```

are all equivalent, though the last is harder to read and thus less desirable.

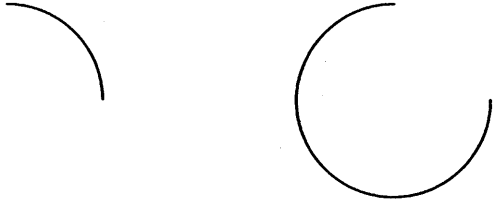
The `from` and `to` attributes are particularly useful with arcs, to specify the endpoints. By default, arcs are drawn counterclockwise,

```
arc from 0.5i,0 to 0,0.5i
```

is the short arc and

```
arc from 0,0.5i to 0.5i,0
```

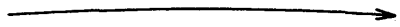
is the long one:



If the `from` attribute is omitted, the arc starts where you are now and goes to the point given by `to`. The radius can be made large to provide flat arcs:

```
arc -> cw from 0,0 to 2i,0 rad 15i
```

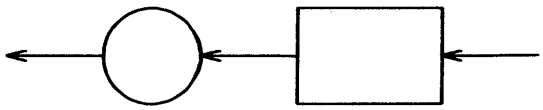
produces



We said earlier that objects are normally connected left to right. This is an over-simplification. The truth is that objects are connected together in the direction specified by the most recent `up`, `down`, `left` or `right` (either alone or as part of some object). Thus, in

```
arrow left; box; arrow; circle; arrow
```

the `left` implies connection towards the left:



This could also be written as

```
left; arrow; box; arrow; circle; arrow
```

Objects are joined in the order determined by the last `up`, `down`, etc., with the entry point of the second object attached to the exit point of the first. Entry and exit points for boxes, circles and ellipses are on opposite sides, and the start and end of lines, motions and arcs. It's not entirely clear that this automatic connection and direction selection is the right design, but it seems to simplify many examples.

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 it was when entered. Nothing else is restored. There is also a more general way to group objects, using `[` and `]`, which is discussed in a later section.

2.4 Labels and Corners

Objects can be labelled or named so that you can talk about them later.

For example,

```
.PS
Box1:
     box ...
     # ... other stuff ...
     move to Box1
.PE
```

Place names have to begin with an upper case letter (to distinguish them from variables, which begin with lower case letters). The name refers to the "center" of the object, which is the geometric center for most things. It's the beginning for lines and motions.

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   # same as previous
line to Box1 - 0.5,0
```

The reserved name `Here` may be used to record the current position of some object, for example as

```
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.

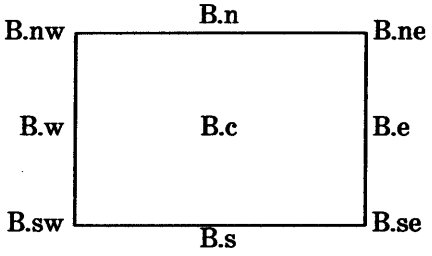
You can also refer to previously drawn objects of each type, using the word `last`. For example, given the input

```
box "A"; circle "B"; box "C"
```

then '`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.

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 box above was produced with

```
.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 center, a start, and an end.) There are a host of (i.e., too many) ways to talk about 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
```

A longer statement like

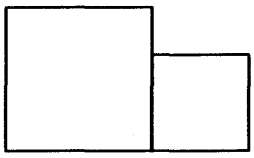
```
line from upper left of 2nd last box to bottom of 3rd last ellipse
```

begins to wear after a while, but it is descriptive.

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

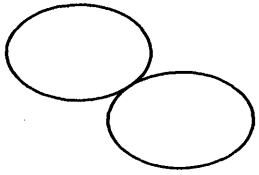


Notice that the corner after `with` is written `.sw`.

As another example, consider

```
ellipse; ellipse with .nw at last ellipse.se
```

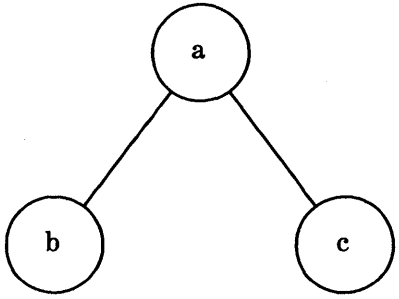
which makes



Sometimes it is desirable to have a line intersect a circle at a point which is not one of the eight compass points that **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
```

produces



By default the line is chopped by **circlerad** at each end. This may be changed:

```
line ... chop r
```

chops both ends by *r*, and

```
line ... chop r1 chop r2
```

chops the beginning by *r1* and the end by *r2*.

There is one other form of positioning that is sometimes useful, to refer to a point some fraction of the way between two other points. This can be expressed in **pic** as

fraction `of the way between` *position1* `and` *position2*

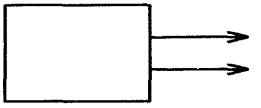
fraction is any expression, and *position1* and *position2* are any positions. You can abbreviate this phrase; "of the way" is optional, and the whole thing can be written instead as

fraction `<` *position1* `,` *position2* `>`

As an 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



Naturally, the distance given by *fraction* can be greater than 1 or less than 0.

2.5 Variables and Expressions

It's generally a bad idea to write everything in absolute coordinates if you are likely to change things. **pic** variables let you parameterize 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 may use the standard operators `+`, `-`, `*`, `/`, and `%`, and parentheses for grouping.

Probably the most important variables are the predefined ones for controlling the default sizes of objects, listed in Section 3. 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 NE corner of Box1
Box1.wid        # the width of Box1
Box1.ht         # and its height
2nd last circle.rad # the radius of the 2nd last circle
```

Any pair of expressions enclosed in parentheses defines a position; furthermore such positions can be added or subtracted to yield new positions:

$(x_1,y_1)+(x_2,y_2)$

are positions, then

(p_1,p_2)

refers to the point

$(p_{1.x},p_{2.y})$

2.6 More on Text

Normally, text is centered at the geometric center of the object it is associated with. The attribute `ljust` causes the left end to be at the specified point (which means that the text lies to the right of the specified place!), and `rjust` puts the right end at the place. `above` and `below` center the text one half line space in the given direction.

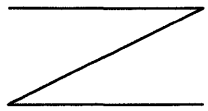
At the moment you can *not* compound text attributes. It is illegal to say `"..." above ljust`.

Text is most often an attribute of some other object, but you can also have self-standing text:

```
"this is some text" at 1,2 ljust
```

2.7 Lines and Splines

A "line" may actually be a path, that is, it may consist of connected segments like this:



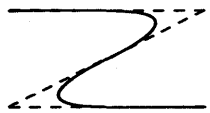
This line was produced by

```
line right 1i then down .5i left 1i then right 1i
```

A spline is a smooth curve guided by a set of straight lines just like the line above. It begins at the same place, ends at the same place, and in between is tangent to the mid-point of each guiding line. The syntax for a spline is identical to a (path) line except for using `spline` instead of `line`. Thus:

```
line dashed right 1i then down .5i left 1i then right 1i
spline from start of last line \
  right 1i then down .5i left 1i then right 1i
```

produces



(Long input lines can be split by ending each piece with a backslash.)

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`, etc. If necessary to disambiguate, the word `then` can be used to separate components, as in

```
spline right then up then left then up
```

which is not the same as

```
spline right up left up
```

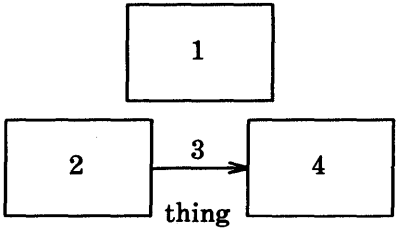
At the moment, arrowheads may only be put on the ends of a line or spline; splines may not be dotted or dashed.

2.8 Blocks

Any sequence of **pic** statements may be enclosed in brackets `[ ... ]` to form a block, which can then be treated as a single object, and manipulated rather like an ordinary box. For example, if we say

```
box "1"
[ box "2"; arrow "3" above; box "4" ] with .n at last box.s - (0,0.1)
"thing" at last [].s
```

we get



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 []`, etc., just like `last box`.

Blocks have the same compass corners as boxes (determined by the bounding box). It is also possible to 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 things like boxes do (i.e., 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

```
last [].A
```

or

```
B.A
```

where `B` is a name attached to a block like so:

```
B : [ ... ;  A: ...;  ]
```

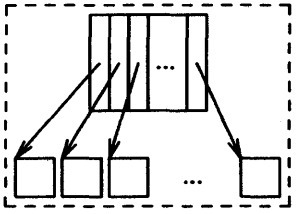
When combined with **`define`** statements (next section), blocks provide a reasonable simulation of a procedure mechanism.

Although blocks nest, it is currently possible to look only one level deep with constructs like `B.A`, although `A` may be further qualified (i.e., `B.A.sw` or `top of B.A` are legal).

The following example illustrates most of the points made above about how blocks work.

```
h = .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 []
```

This produces



2.9 Macros

Pic provides a rudimentary macro facility, the simple form of which is identical to that in **eqn**:

```
define   name  X  replacement text  X
```

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, ...)
```

Non-existent arguments are replaced by null strings.

As an example, one might define a `square` by

```
define square X box ht $1 wid $1  $2 X
```

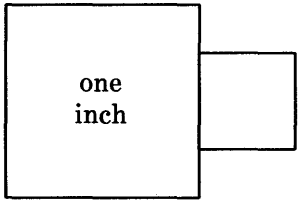
Then

```
square(1i, "one" "inch")
```

calls for a one-inch square with the obvious label, and

```
square(0.5i)
```

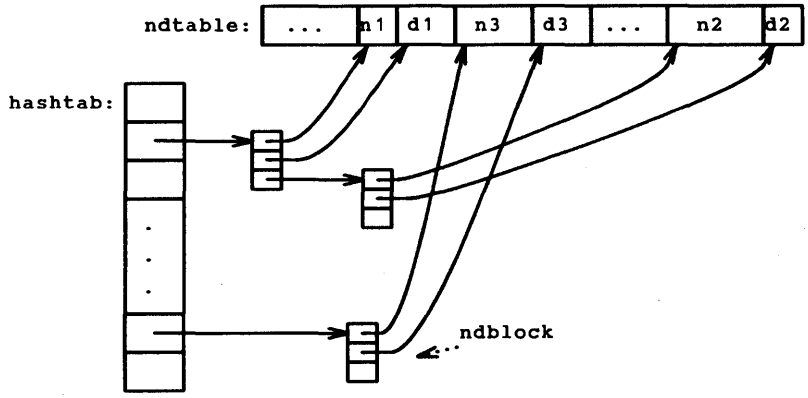
calls for a square with no label:



Coordinates like *x,y* may be enclosed in parentheses, as in (*x,y*), so they can be included in a macro argument.

2.10 Some Examples

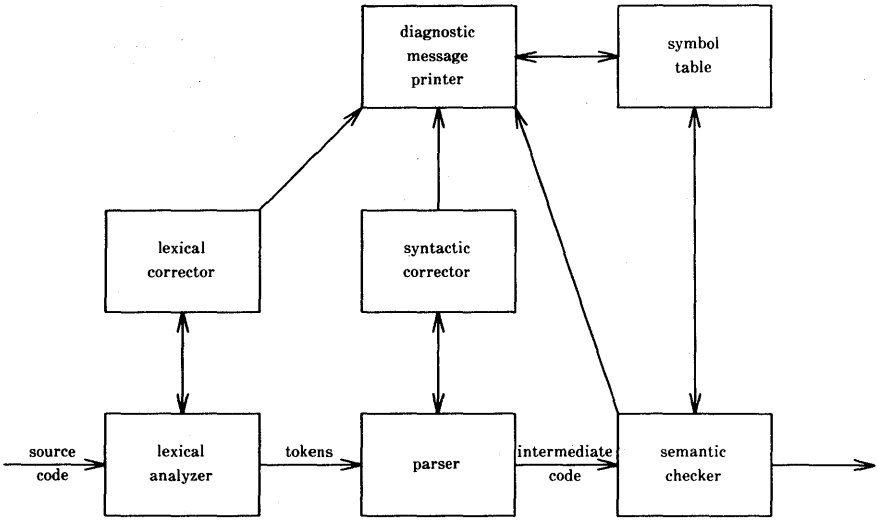
Here are a few larger examples:



The input for the picture above was:

```
define ndblock X
      box wid boxwid/2 ht boxht/2
      down;  box same with .t at bottom of last box;   box same
X
boxht = .2i; boxwid = .3i; circlerad = .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"
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)
circle invis "ndblock"  at last box.e + (.7i,.2i)
arrow dotted from last circle to last box chop
box invis wid 2*boxwid "ndtable:" with .e at Start.w
```

The second example follows.



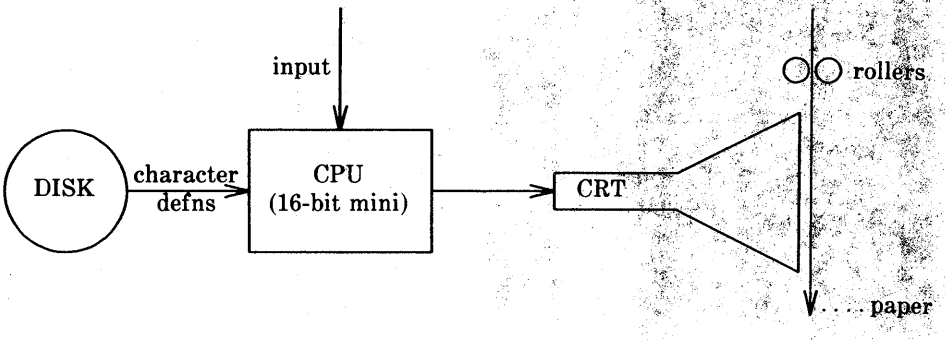
This input will generate a picture something like the above:

```
.PS 6
.ps 8
     arrow "source" "code"
LA:  box "lexical" "analyzer"
     arrow "tokens" above
P:   box "parser"
     arrow "intermediate" "code"
Sem: box "semantic" "checker"
     arrow

     arrow <-> up from top of LA
LC:  box "lexical" "corrector"
     arrow <-> up from top of P
Syn: box "syntactic" "corrector"
     arrow up
DMP: box "diagnostic" "message" "printer"
     arrow <-> right  from right of DMP
ST:  box "symbol" "table"
     arrow from LC.ne to DMP.sw
     arrow from Sem.nw to DMP.se
     arrow <-> from Sem.top to ST.bot
.PE
```

There are eighteen objects (boxes and arrows) in the second example, and one line of **pic** input for each; this seems like an acceptable level of verbosity.

The next example is the following:



Basic Digital Typesetter

This input will generate a picture like that in example 3:

```
.PS 5
circle "DISK"
arrow "character" "defns"
box "CPU" "(16-bit mini)"
{ arrow <- from top of last box up "input " rjust }
arrow
CRT: "   CRT" ljust
line from CRT - 0,0.075 up 0.15 \
then right 0.5 \
then right 0.5 up 0.25 \
then down 0.5+0.15 \
then left 0.5 up 0.25 \
then left 0.5

Paper: CRT + 1.0+0.05,0
arrow from Paper + 0,0.75 to Paper - 0,0.5
{ move to start of last arrow down 0.25
  { move left 0.015; circle rad 0.05 }
  { move right 0.015; circle rad 0.05; "   rollers" ljust }
}
"  paper" ljust at end of last arrow right 0.25 up 0.25
line left 0.2 dotted
.PE
.ce
Basic Digital Typesetter
```

3. PIC Reference Manual

3.1 Pictures

The top-level object in **pic** is the "picture":

picture:
```
.PS optional-width
element-list
.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 instead the line is

```
.PS <f
```

the file `f` is inserted in place of the `.PS` line.

If `.PF` is used instead of `.PE`, the position after printing is restored to what it was upon entry.

3.2 Elements

An *element-list* is a list of elements. The elements are

element:
```
primitive attribute-list
placename : element
placename : position
variable = expression
direction
troff-command
{ element-list }
[ element-list ]
```

Elements in a list 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.

Variable names begin with a lower case letter; place names begin with upper case. Place and variable names retain their values from one picture to the next.

The current position and direction of motion are saved upon entry to a { . . . } block and restored upon exit.

Elements within a block enclosed in [. . .] are treated as a unit; the dimensions are determined by the extreme points of the contained objects. Names, variables, and direction of motion within a block are local to that block.

The *troff-command* is any line that begins with a period. Such lines are assumed to make sense in the context where they appear.

3.3 Primitives

The primitive objects are

primitive:
```
box
circle
ellipse
arc
line
arrow
move
spline
"any text at all"
```

`arrow` is a synonym for `line ->`.

3.4 Attributes

An *attribute-list* is a sequence of zero or more attributes; each attribute consists of a keyword, perhaps followed by a value. In the following, *e* is an expression and *opt-e* an optional expression.

attribute:

`h(eigh)t` *e*	`wid(th)` *e*
`rad(ius)` *e*	`diam(eter)` *e*
`up` *opt-e*	`down` *opt-e*
`right` *opt-e*	`left` *opt-e*
`from` *position*	`to` *position*
`at` *position*	`with` *corner*
`by` *e, e*	`then`
`dotted` *opt-e*	`dashed` *opt-e*
`chop` *opt-e*	`->` `<-` `<->`
`same`	`invis`
text-list	

Missing attributes and values are filled in from defaults. Not all attributes make sense for all primitives; irrelevant ones are silently ignored.

These are the currently meaningful attributes:

```
box:
     height, width, at, dotted, dashed, invis, same, text
circle and ellipse:
     radius, diameter, height, width, at, invis, same, text
arc:
     up, down, left, right, height, width, from, to, at, radius,
     invis, same, cw, <-, ->, <->, text
line,arrow
     up, down, left, right, height, width, from, to, by, then,
     dotted, dashed, invis, same, <-, ->, <->, text
spline:
     up, down, left, right, height, width, from, to, by, then,
     invis, <-, ->, <->, text
move:
     up, down, left, right, to, by, same, text
"text...":
     at, text
```

The attribute **at** implies placing the geometrical center at the specified place. For lines, splines and arcs, **height** and **width** refer to arrowhead size.

3.5 Text

Text is normally an attribute of some primitive; by default it is placed at the geometrical center of the object. Stand-alone text is also permitted. A *text-list* is a list of text items; a text item is a quoted string optionally followed by a positioning request:

```
text-item:
     "..."
     "..." center
     "..." ljust
     "..." rjust
     "..." above
     "..." below
```

If there are multiple text items for some primitive, they are centered vertically except as qualified. Positioning requests apply to each item independently.

Text items can contain **troff** commands for size and font changes, local motions, etc., but make sure that these are balanced so that the entering state is restored before exiting.

3.6 Positions and Places

A position is ultimately an *x,y* coordinate pair, but it may be specified in other ways.

position:

e, e
`place` ± *e, e*
`(` *position, position* `)`
e [`of the way`] `between` *position* `and` *position*
e `<` *position , position* `>`

The pair *e, e* may be enclosed in parentheses.

place:

`placename` *optional-corner*
corner `placename`
`Here`
corner `of` *nth primitive*
nth primitive optional-corner

A *corner* is one of the eight compass points or the center or the start or end of a primitive. (Not text!)

corner:

`.n .e .w .s .ne .se .nw .sw`
`.t .b .r .l`
`.c .start .end`

Each object in a picture has an ordinal number; *nth* refers to this.

nth:

n`th`
n`th last`

Legal input includes `1th`, as well as synonyms like `1st` and `3st`.

3.7 Variables

The built-in variables and their default values are:

```
boxwid 0.75i                    boxht 0.5i
circlerad 0.25i
ellipsewid 0.75i                ellipseht 0.5i
arcrad 0.25i
linewid 0.5i                    lineht 0.5i
movewid 0.5i                    movewid 0.5i
arrowht 0.1i                    arrowwid 0.05i
dashwid 0.1i
scale 1
```

These may be changed at any time, and the new values remain in force until changed again. Dimensions are divided by `scale` during output.

3.8 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
place .rad

3.9 Definitions

The **define** statement is not part of the grammar.

define:
    define *name* **X** *replacement text* **X**

Occurrences of **$1** through **$9** in the replacement text will be replaced by the corresponding arguments if **name** is invoked as

name(*arg1, arg2, ...* **)**

Non-existent arguments are replaced by null strings. *Replacement text* may contain newlines.

Chapter 4

MATHEMATICS TYPESETTING PROGRAM

PAGE

Chapter 4

MATHEMATICS TYPESETTING PROGRAM

1. Introduction

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

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

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

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

The **eqn** software for typesetting mathematics has been designed to be easy to learn and to use by people (for example, secretaries and mathematical typists) who know neither mathematics nor typesetting. The language can be learned in an hour or so since it has few rules and fewer exceptions. It interfaces directly with the phototypesetting language so mathematical expressions can be embedded in the running text of a manuscript, and the entire document produced in one process. Typical mathematical expressions include size and font changes, positioning, line drawing, and other

necessary functions to print according to mathematical conventions, and are done automatically. The syntax of the language is specified by a small context-free grammar; a compiler-compiler is used to make a compiler that translates this language into typesetting commands. Output may be produced on either a typesetter or on a terminal with forward and reverse half-line motions.

2. Usage

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

To typeset mathematical text stored in *files*, the following command is issued:

eqn *files* | troff

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

eqn *files* | troff -mm

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

To use a specific terminal as the output device, the following command is used:

neqn *files* ¦ nroff -T*x*

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

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

tbl *files* ¦ eqn ¦ troff
tbl *files* ¦ neqn ¦ nroff

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

3. Language

3.1 Design

The fundamental principle upon which the **eqn** language design is based is that the language should be easy to use by those who know neither mathematics nor typesetting. This principle implies:

- Normal mathematical conventions about operator precedence, such as parentheses, cannot be used. To give special meaning to such characters means that the user has to understand what is being typed. The language should not assume that parentheses are always balanced.

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

- Standard things should happen automatically. When "x=y+z+1" is typed, "x=y+z+1" should be the result. Subscripts and superscripts should be printed automatically (with no special intervention) in appropriately smaller size. Fraction bars should be made the right length and positioned at the correct height. A mechanism for overriding default actions should exist, but its application is the exception, not the rule.

A secondary, but still important, design goal is that the system should be easy to build and to change. To this end and to guarantee regularity, the language is defined by a context-free grammar.

The typist should have a reasonable picture (a 2-dimensional representation) of the desired final form, such as might be handwritten by the author of a paper. It is also assumed that the input is to be typed on a computer terminal much like an ordinary typewriter. This implies an input alphabet of perhaps 100 characters, none of them special.

The **troff** processor performs work for the mathematics typesetting function. It is a powerful program, with a macro facility, text and arithmetic variables, numerical computation and testing, and conditional branching. Text strings are passed to the **troff** formatter omitting the need for a separate storage management package. The user need not be concerned with most details of the particular device and character set currently in use. For example, the **troff** formatter computes the widths of all strings of characters; the user does not need to know about them.

3.2 Structure

The basic structure of the language is not original. Equations are pictured as a set of boxes, pieced together in various ways. For example, something with a subscript is a box followed by another box moved downward and shrunk an appropriate amount. A fraction is a box centered above another box, at the right altitude, with a line of correct length drawn between them.

3.3 Mode of Operation

Since the **eqn** program is useful for typesetting mathematics only, it interfaces with the underlying typesetting language in order to get intermingled mathematics and text. The standard mode of operation is that when a document is typed, mathematical expressions are input as part of the text but marked by delimiters, **.EQ** and **.EN**. The program reads this input and treats as comments those things which are not mathematics passing them through untouched. At the same time, it converts mathematical inputs into **troff** formatter commands. The resulting output is passed directly to the formatter where comments and mathematical parts become text and/or formatter commands.

4. User's Guide

4.1 Delimiters

The **eqn** preprocessor reads intermixed text and equations and passes its output to the **troff** formatter. Since the formatter uses lines beginning with a period as control words (**.ce** means "center the next output line"), **eqn** uses the **.EQ** macro to mark the beginning of an equation and the **.EN** macro to mark the end. By default **.EQ** and **.EN** are ignored by the **troff** formatter, so equations are printed in-line.

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

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

The **.EQ** and **.EN** delimiters are passed through to the formatter untouched, so they can be used to center equations, number them automatically, etc. The **troff** and **nroff** formatter macro package, **−mm**, allows equations to be left-justified and numbered. Any argument to the **.EQ** macro will be placed at the right margin as an equation number.

Warning: When using the* -mm *macro package, always use a break-producing request such as* .br *or* .sp *immediately before the* .EQ *macro.

For example, the input

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

produces the output

$$x = f(y/2) + y/2 \qquad (4.1a)$$

Since it is tedious to type **.EQ** and **.EN** around very short expressions (e.g., single letters), two characters can be defined to serve as the left and right delimiters of expressions. These characters are recognized anywhere in subsequent text {4.16}.

4.2 Spaces and New Lines

4.2.1 Input Spaces

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

Thus an input

```
x   =   y
  + z + 1
```

produces

$$x = y + z + 1$$

Free-form input is easier to type initially. Space and newline

characters should be freely used to make input equations readable and easy to edit. Very long lines are hard to correct if a mistake is made.

4.2.2 Output Spaces

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

```
x~=~y~+~z
```

produces

$$x = y + z$$

4.3 Symbols, Special Names, and Greek Alphabet

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

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

produces

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

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

converted to Roman font:

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

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

Spaces should be put around separate parts of the input. A common error is to type "f(pi)" without leaving spaces on both sides of the "pi". As a result, **eqn** does not recognize *pi* as a special word, and it in the output. A list of **eqn** names appears in Figure 4-1.

INPUT NAME	*OUTPUT CHARACTER*
> =	≥
< =	≤
= =	≡
! =	≠
+ -	±
- >	→
< -	←
<<	≪
>>	≫
inf	∞
partial	∂
half	½
prime	′
approx	≈
nothing	
cdot	·
times	×
del	∇
grad	∇
...	. . .
,....,	, . . . ,
sum	Σ
int	∫
prod	Π
union	∪
inter	∩
DELTA	Δ
GAMMA	Γ
LAMBDA	Λ
OMEGA	Ω

Figure 4-1. Names Recognized by eqn (Sheet 1 of 2)

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

Figure 4-2. Names Recognized by eqn (Sheet 2 of 2)

Four-character **troff** names can also be used for anything **eqn** does not recognize, e.g., "\(pl" for the + sign.

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

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

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

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

4.4 Subscripts and Superscripts

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

$$x^2+y_k$$

is produced by

```
x sup 2 + y sub k
```

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

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

```
x sub i sub 1
```

produce

x_{i_1}

A subscript and superscript on the same thing are printed one above the other if the subscript comes first.

```
x sub i sup 2
```

is

x_i^2

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

```
x sup y sub z
```

generates

x^{y_z}

not

$x^{y}{}_{z}$

A common erroneous expression is of the form

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

which causes

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

instead of the intended

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

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

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

4.5 Braces

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

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

```
e sup {i omega t}
```

produces

$$e^{i\omega t}$$

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

Braces can occur within braces if necessary. The statement

```
e sup {i pi sup {rho +1}}
```

generates

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

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

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

4.6 Fractions

Fractions are specified with the keyword *over*.

```
a+b over c+d+e = 1
```

produces

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

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

```
-b sup 2 over pi
```

is

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

4.7 Square Roots

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

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

yields

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

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

4.8 Summations, Integrals, and Similar Constructions

Summations, integrals, and similar constructions are easy.

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

produces

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

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

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

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

```
int
prod
union
inter
```

which become, respectively

$$\int \prod \bigcup \bigcap$$

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

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

is

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

4.9 Size and Font Changes

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

```
bold x y
```

is

x*y*

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

bold {x y} z

produces

xyz

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

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

A = fat {pi r sup 2}

produces

$A = \boldsymbol{\pi r^2}$

Legal sizes which may follow *size* are

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

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

size +2

makes the size two points larger. This has the advantage that

knowledge of the current size is not necessary.

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

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

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

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

4.10 Diacritical Marks

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

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

The diacritical mark is placed at the correct height, and *bar* and *under* are made the right length for the entire construct. Other

marks are centered. An example of an expression using diacritical marks is:

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

It is made by typing

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

4.11 Quoted Text

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

```
italic " sin(x)"  + sin (x)
```

produces

$$sin(x) + \sin(x)$$

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

```
" { size alpha } "
```

prints

{ *size alpha* }

and

```
roman " { size alpha }"
```

prints

```
{ size alpha }
```

The "" construction is often used as a place-holder when grammatically **eqn** needs something, but nothing is actually wanted on the output.

4.12 Aligning Equations

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

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

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

produces

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

The *mark* and *lineup* operations do not work with centered equations. Also, *mark* does not look ahead.

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

is not going to work because there is not room for the $x+y$ part after the *mark* remembers where the x is.

4.13 Big Brackets

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

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

produces

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

The resulting brackets are made large enough to cover whatever they enclose. Other characters can be used besides these, but they are not likely to look very good. One exception is the *floor* and *ceiling* characters.

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

produces

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

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

The *right* keyword may be omitted. A "left something" need not have a corresponding "right something". If the right part is omitted, braces are put around the thing that the left bracket is to encompass. Otherwise, resulting brackets may be too large. If the left part is to be omitted, things are more complicated because technically a *right*

cannot exist without a corresponding *left*. Instead the following input will do:

```
left " " ... right)
```

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

4.14 Piles

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

Three other forms of pile exist:

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

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

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

the following is input.

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

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

4.15 Matrices

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

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

the following text is the input:

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

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

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

Note: Each column must have the same number of elements.

4.16 In-Line Equations

In a mathematical document, it is necessary to follow mathematical conventions in display equations and in text. Making variable names (such as *x*) italic is one instance. Although this could be done by surrounding the appropriate parts with **.EQ** and **.EN**, the continual repetition of **.EQ** and **.EN** is a nuisance. Furthermore, with **−mm**, **.EQ** and **.EN** imply a displayed equation.

The **eqn** program provides a shorthand notation for short in-line equations. Two characters can be defined to mark the left and right ends of an in-line equation, and then expressions can be typed in the middle of text lines.

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

The three lines added to the beginning of the document set both the left and right delimiter characters to dollar signs. A sample input is:

```
Let $alpha sub i$ be the primary variable, and let $beta$
be zero. Then it can be shown that $x sub 1$ is $>=0$.
```

to produce:

> Let α_i be the primary variable, and let β be zero. Then it can be shown that x_1 is ≥ 0.

This works as expected—space characters, newline characters, etc., are significant in the input text, but not in the resultant equation. Multiple equations can occur in a single input line. Space is left before and after a line that contains in-line expressions so that a tall expression will not interfere with surrounding lines. To turn off the delimiters:

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

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

- Do not use braces, tildes, circumflexes, or double quotes as delimiters.
- In-line font changes must be closed before in-line equations are encountered.

4.17 Defines

There is a definition facility, so a user can say

```
define name '...'
```

at any time in the document. Henceforth, any occurrence of *name* in an expression will be expanded into whatever was inside the quotes in its definition. This lets users tailor the language to their own specifications. For example, if the sequence

```
x sub i sub 1 + y sub i sub 1
```

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

```
define  xy  'x sub i sub 1 + y sub i sub 1'
```

This define makes *xy* a shorthand for whatever characters occur between the single quotes in the definition. Any character can be used instead of the quote to mark the ends of the definition as long as it does not appear inside the definition.

The above expression can now be input as follows:

```
.EQ
f(x) = xy ...
.EN
```

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

Although definitions can use previous definitions, as in:

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

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

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

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

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

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

```
define  /  ' over '
```

To redefine *over* as / use:

```
define  over  ' / '
```

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

4.18 Local Motions

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

4.19 Precedence

Precedence rules resolve the ambiguity in a construction like

```
a sup 2 over b
```

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

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

The following operations group to the left:

```
over sqrt left right
```

All others group to the right.

5. Troubleshooting

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

```
syntax error between lines x and y, file z
```

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

```
eqn files >/dev/null
```

discards the output but prints the message.

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

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

READER COMMENT FORM

UNIX SYSTEM V DOCUMENTER'S WORKBENCH, VOLUME TWO
Macro Package and Preprocessor Reference

Altos Computer Systems
2641 Orchard Parkway
San Jose, CA 95134

This document has been prepared for use with your Altos Computer System. Should you find an error or problem in this manual, please write it down (noting page number), and return this form to the ALTOS PUBLICATIONS DEPARTMENT.

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

System Model Number_____

Serial Number_____

Document Title_____

Revision Number 690-15844-001 Date_____

Name_____

Company Name_____

Address_____

_____

_____

Printed in U.S.A.
P/N 690-15844-001

2641 Orchard Park Way, San Jose, California 95134
(408) 946-6700, Telex 470642 Alto UI

July 1985