Microport System V/AT
Text Preparation System Ver. 2.3
Release and Installation Notes

March 1988

INTRODUCTION

You have received the newest release of the Text Preparation System from *Microport Systems*. The changes to this software have been made to improve the product for our customers.

INSTALLATION DESCRIPTION

This software package is in our "installit" format to make the installation as trouble free as possible. To install the package, simply insert each diskette into the first floppy drive and type "installit".

CONTENT DESCRIPTION

This release contains several programs that were modified to work with "csh". Another program that was previously called "man" is now called "tman". This name change allows the Runtime "man" program to coexist with this different version from the Text system. The Runtime "man" looks in /usr/catman for formatted manual pages. The Text system "tman" looks in /usr/man for "nroff" or "troff" versions of manual pages.

The modified files are the following:

/usr/bin/tman
/usr/bin/mm
/usr/bin/mmt
/usr/bin/spell
/usr/lib/spell/compress
/usr/lib/manprog

If there are any problems concerning this package, please report the details to the *Microport* technical support group immediately.

Text Preparation System

Reorder No. 0103

MICROPORT SYSTEMS

The material contained in this manual was reprinted with permission from AT&T and is comprised of excerpts from the following AT&T manuals.

*UNIX **System V**	April 1984
†DOCUMENTER'S WORKBENCH **Software** 1.0	307-150, Issue 2
Introduction and Reference Manual	

UNIX **System V**	
DOCUMENTER'S WORKBENCH **Software** 1.0	April 1984
Text Formatters Reference	307-151, Issue 2

UNIX **System V**	
DOCUMENTER'S WORKBENCH **Software** 1.0	April 1984
Macro Package Reference	307-152, Issue 2

UNIX **System V**	
DOCUMENTER'S WORKBENCH **Software** 1.0	April 1984
Preprocessor Reference	307-153, Issue 2

QUICK REFERENCE GUIDE

Table of Contents

Chapter 1

DOCUMENT PREPARATION

INTRODUCTION

This chapter introduces you to the process of producing a document using the DOCUMENTER'S WORKBENCH software on the UNIX system. The three major tasks involved in document preparation are:

- Inputting
- Formatting
- Printing or typesetting.

Entering data into a computer system is known as inputting. Inputting is normally done by using a keyboard and an editing program. The data stored in a file for document preparation consists of text and text formatting commands that the text formatter understands. Before the data can be printed, the formatting commands must be converted by a text formatter to information a printer or phototypesetter understands. This process is called formatting. There are several formatting programs that prepare the data to be printed.

Everything from how to get started to producing a final copy is covered in this chapter. Only the basic concepts of the facilities available and how they fit together are described. This chapter does not cover enough of the details for you to be able to write a document after reading it.

This chapter provides a foundation so that you can understand these details later.

Why use the DOCUMENTER'S WORKBENCH software to produce a document? Simply because documents can be produced faster using it. Assume that you are going to produce a large document using tools found in the old office environment. The old way is to spend much time planning the appearance of the following items:

- Page headers
- Titles
- Headings
- Margins
- Lists
- Displays
- Footnotes
- Page Numbering
- Table of Contents
- Glossary
- Index.

For your job, you may need more or fewer items than listed.

You are concerned with how much white space is involved. How much white space goes above, below, to the right, and to the left of the text

that makes up each item?

If you are lucky, the type of document you are trying to produce already has a set of standards associated with it. If not, make your own standards as you go. The DOCUMENTER'S WORKBENCH software contains some built-in standards that are implemented by default. These built-in default values can be tailored to meet your specific needs.

Assume that you have produced your document on a typewriter. To correct a single error, add text, delete text, or change text could mean you must retype a page or part of the entire document. Most documents go through several versions before they are finally finished. Therefore, much of your time would be spent in planning white space and physically manipulating the words of your document. This process takes away valuable time that could be spent on the content of your text.

The DOCUMENTER'S WORKBENCH software does not eliminate white space planning or manipulating words. It does speed up these processes significantly. Writing and editing becomes faster, more systematic, and more enjoyable. This software enables you to create, change, display, and print text by a method more efficient than found in the old office environment. It also allows you to concentrate more on document content and less on document appearance by automatically controlling appearance.

GETTING STARTED

The first thing a beginning user should remember is don't get discouraged if your document doesn't look as you planned. Usually, this occurs when you don't fully understand the tools that are available to you. It takes time to develop your skills. Experiment with the software. Try different things just to see what the result will be. Then refer to the manual again. Usually this will give you a better understanding of the particular software tool you are using.

Some of the things you experience during the learning period are discussed in this chapter. Your goal is to learn how to produce letters, reports, memorandums, manuals, books, or some type of

document. You need to grow daily in editing, document processing, and even some simple programming. You will evolve from experimenting with the software to producing documents from initial start through final copy. As you grow in your knowledge, you will learn how the tools provided assist you in your day-to-day job.

INPUTTING YOUR DOCUMENT

1. Text

The DOCUMENTER'S WORKBENCH software requires input text to contain instructions for the appearance you want. This text is prepared by you using a text editor such as ed(1) or vi(1) (see section 1 in the Runtime System manual).

The input is composed of the text you want printed as well as formatter requests which tell the program how you want your text to look on the page.

Total input therefore consists of text lines that are printed, mixed with control lines that control format. These control lines are interpreted by a text processor and do not appear in the formatted output. Text is controlled in document preparation by the following features:

- Requests
- Macros
- Strings
- Registers.

2. Requests

A request is a control line embedded in your text. A text processing request has the following characteristics:

- The request must be entered on a line by itself.
- The request must start with a period (.) or an acute accent (') at the beginning of the line, followed by two lowercase characters such as (.xx).
- The request can be located anywhere in the document but normally affects only the text that follows it.

- The request can have optional arguments located on the same line.

Your entire document consists of text interspersed with requests. Writing your document with requests is a lot like writing a program in a low level language.

Your document could contain thousands of paragraphs. Assume you need every new paragraph to start with some vertical space and be indented. The following **nroff/troff** requests could be written to achieve this:

```
.sp 1              \" space one line
.ti +5   \" temporarily indent 5 spaces
```

To show you how complex a set of requests can be, the following list is included. This list of requests also produces a paragraph.

```
.if!(((\\n(!D=\\n(nl):(\\n(!D=(\\n(nl-.5v)))&\
(\\n(!Z=\\n(.k)&(\\n(Np=0)) \{\
.br
.nr;1 \\n(:J
.nr;2 \\n(nl
.sp \\n(Psu*.5v
.if!\\n(:D .ne 1v+.5p
.ie!\\n(;1-\\n(:J .nr ;2 \\n(;2-\\n(:J
.el.nr ;2 \\n(nl-\\n(:J
.nr:J \\n(;2
.if\\n(.$>0&(0\\$1) .ti+\\n(Pin
.if\\n(.$=0 \{\
.if\\n(Pt=1 .ti+\\n(Pin
.if\\n(Pt>1&(\\n(:I) .ti+\\n(Pin
.if\\n(Pt>1&(\\n(:I=0)&(\\n(:J>0) .ti+\\n(Pin \}
.if\\n(Np \{\
\\n(H1.\\n+(!4\ \ \c
'br\}
.nr:I 1 \}
.nr:u  0
```

As you can see, this is very complicated. These requests are used by the memorandum macro package to define a paragraph. The reason for such complexity is that several decisions are made concerning spacing, indent, and numbering, based on the values of formatter variables. These variables are in the form of number registers (Pi, Pt, :J, Ps, Np) and character strings ($1). These will be discussed in more detail later.

How would you like to type in a long collection of requests just to begin a paragraph? Is there some way to avoid this? Yes, a macro can condense a collection of requests into one command line.

3. Macros

A macro is a 1- or 2-character abbreviation (name) that is replaced by a sequence of formatting requests when processed. For example, the long collection of requests to produce spacing and indentation for a paragraph can be replaced with

```
.P
```

on a line by itself. This is done by defining the **P** macro as follows:

```
.de P
.if!(((\\n(!D=\\n(nl):(\\n(!D=(\\n(nl-.5v)))&\
(\\n(!Z=\\n(.k)&(\\n(Np=0)) \{\
.br
.nr;1 \\n(:J
.nr;2 \\n(nl
.sp \\n(Psu*.5v
.if!\\n(:D .ne 1v+.5p
.ie!\\n(;1-\\n(:J .nr ;2 \\n(;2-\\n(:J
.el.nr ;2 \\n(nl-\\n(:J
.nr:J \\n(;2
.if\\n(.$>0&(0\\$1) .ti+\\n(Pin
.if\\n(.$=0 \{\
.if\\n(Pt=1 .ti+\\n(Pin
.if\\n(Pt>1&(\\n(:I) .ti+\\n(Pin
.if\\n(Pt>1&(\\n(:I=0)&(\\n(:J>0).ti+\\n(Pin \}
.if\\n(Np \{\
\\n(H1.\\n+(!4\ \ \c
'br\}
.nr:I 1 \}
.nr:u  0
..
```

This is the paragraph macro from the memorandum macro package that was shown earlier. Notice that the definition of the macro begins with ".de" and ends in "..". Thus by issuing one simple macro request, several formatting requests can be invoked. Macros and formatting requests can be interspersed in the same document.

A macro can be predefined by some existing macro package or you can define your own macros. When preparing documents that repeat the same text processing primitive requests over and over, it becomes desirable to define your own macro. Some rules for defining and using macros follow:

1. Macro names are one or two characters that should not be the same as any text processing requests. User-defined macro names are normally made uppercase.

2. The **.de** request begins a macro definition. This request is on a line by itself and has the following form:

.de XX

where XX is the macro name.

3. The list of text processing requests to be repeated follows the start of the macro definition. These requests make up the macro function.

4. The macro definition is ended with a double period (..) on a line by itself.

5. Up to nine arguments can be passed to a macro. Each argument is separated from other arguments by a space. When you invoke the macro in your input, the form is as follows:

.XX arg1 arg2 arg3 ... arg9

where XX is the macro name and "arg" are possible arguments. A single argument can contain a space only if it is surrounded by double quotes.

6. The macro definition must appear before its invocation. It is a good practice to place macro definitions at the beginning of your text file or in a separate file that is always called before your text files.

7. Macros may be nested. This means that other macros may be defined or invoked within a macro.

4. Macro Packages

Text processing requests are hard to use effectively. It is very difficult for most beginners to define their own macros. Therefore, several "packages" of predefined formatting macros are provided. These macro packages can be used to create displays, headers, headings, paragraphs, titles, footnotes, listings, multicolumn output, and most of the other items needed to produce a document. You can learn how to use these macro packages with little effort compared to learning how to use formatting requests or defining your own macros. These packages take some effort to learn, but the rewards for using them are so great that it is time well spent.

Macro packages supported are as follows:

- Memorandum macros (mm)
- Sroff/mm
- Macro package for viewgraphs and slides (mv)

These macro packages are more fully explained in the "Memorandum Macros User Guide" and the "Viewgraph Macros User Guide" chapters of this manual.

The memorandum macro package is the standard general purpose package of text formatting macros used with the **nroff, otroff**, and **troff** text formatters to produce many common types of documents. A brief description of the " memorandum macros" package known as **mm** follows. Formatting macro requests typically consist of a period and two uppercase letters, such as

.PH

that is used to define a page header or

.P

to begin a new paragraph.

The text of a typical document is entered so that it looks something like this:

```
.TL
title
.AU "author information"
.MT "memorandum type"
.P
Enter text ---
---
.P
More text ---
---
.SG "signature"
```

The lines that begin with a period are the formatting macro requests. For example, **.P** calls for starting a new paragraph. The precise meaning of **.P** depends on the output device being used (typesetter or terminal, for instance) and the publication the document will appear in. For example, the memorandum macro package normally assumes that a paragraph is preceded by a space—one line in **nroff** and one-half vertical space in **otroff** or **troff**, and that the first line is left justified. These rules can be changed if desired, but they are changed by changing the interpretation of **.P**, not by retyping the document.

The **sroff/mm** package is a smaller subset of the nroff/mm package plus a few additional macros. The **sroff/mm** package is used with the **sroff** text processor to produce documents.

The **mv** macro package is designed to format your text into viewgraphs and slides. The **mv** package is used in conjunction with **otroff** or **troff** to produce phototypeset output. This output can be photographed and made into slides or copied for making transparencies.

5. Strings

A string is a 1- or 2-character variable that is embedded in text or in a macro. It is actually a text register that can be defined to contain a string of characters that can be printed simply by calling the string name. A string may be predefined by a macro package or you may define your own strings. A text string is a group of characters that may be more than one word. Sequences of words or names that occur repeatedly in a document can be replaced by a string. The contents of

a string is printed by entering *x for a single character string name (x), or *(xx for a two character string name (xx). For example, assume that you want to print the current date in your document. If the string **DT** contains the current date, then rather than entering:

The date is December 23, 1923

and updating your file when the date changes, it is simpler to enter:

The date is *(DT

Unless redefined, the predefined string **DT** in the memorandum macro package contains the current date. Strings are often used in page headers, page footers, and lists.

The rules for defining strings are as follows:

1. A string name consists of one or two uppercase characters that should not be the same as a text processing request or macro.

2. The **.ds** request defines a string. A string definition has the following form:

   .ds XX text string

   where XX is the string name.

3. The " text string" can be of any length and can include concealed new lines. [To "conceal" a "newline", precede it with a backslash (\)].

An example of some string definitions follow:

```
.ds UG \fIUNIX System User Reference Manual\fR
.ds U UNIX operating system
```

Now that you know how to define a string, how do you use it? The rules for using strings follow.

1. To use a string in regular text, precede the name by "*" (for a 1-character name) or "*(" (for a 2-character name).

2. To use a string in a macro definition, precede the name with "*" (for a 1-character name) or "*(" (for a 2-character name).

3. Strings may be used anywhere in the text or in a macro.

4. The string must be defined before it can be used.

5. Strings may be nested.

An example of using strings in text follows. The input

```
The \*U
user commands are described in the
\*(UG.
```

results in the following output when formatted with the previous string definitions of **U** and **UG**:

```
The UNIX operating system
user commands are described in the
UNIX System User Reference Manual.
```

6. Registers

Text processors provide three different kinds of registers:

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

The predefined registers have default values. These registers are maintained by the text formatters. They are used to define part of the overall appearance of your document. A general number register can be read, written, automatically incremented or decremented, and

interpolated into the input. Number registers may also be used in numerical expressions (arithmetic), for flags, and for automatic numbering.

In the memorandum macro package, the appearance of a paragraph is controlled by the following registers:

- **Pi**
- Pt
- Ps

A register can be given a value using the **.nr** text processing request. For example, to indent paragraphs by three spaces, the paragraph indent register (**Pi**) is set to 3 and the paragraph type register (**Pt**) is set to 1 (for all paragraphs indented) at the beginning of your document:

```
.nr Pi 3
.nr Pt 1
```

The initial values of **Pt** and **Pt** are 0, which causes paragraphs to be left justified.

By default, the amount of space between paragraphs is one blank line. The **Ps** number register controls spacing between paragraphs. To force two blank lines (double spacing) between paragraphs, the following would be used:

```
.nr Ps 2
```

These and other registers used to control format by the memorandum macro package are more fully explained in the "Memorandum Macros User Guide" and the "Viewgraph Macros User Guide" chapters of this manual.

To interpolate the value of a number register into the input, simply prepend a backslash **n** to the register name. For instance, the input:

```
Paragraphs are separated by \n(Ps
blank lines in this chapter.
```

would result in the following output when formatted:

```
Paragraphs are separated by 3
blank lines in this chapter.
```

The rules for defining number registers follow:

1. The name is one or two characters and is not the same as any other number register name. An easy way to avoid conflict with names already used by the formatters and **mm** is to use two letters, the first being lowercase and the second being uppercase.

2. The **.nr** request defines number registers. A number register definition has the following form:

   ```
   .nr R N M
   ```

   where R is the register name, N is the initial value of the register, and M is the amount to automatically increment or decrement the register.

3. The **.af** request sets the format for output of the number register and has the following form:

   ```
   .af R c
   ```

   where R is register name and c defines output format to be arabic, arabic with leading zeros, lowercase roman, uppercase roman, lowercase alphabetic, or uppercase alphabetic as shown for example,

c	Numbering Format
1	0,1,2,3, ...
001	000,001,002,003, ...
i	0,i,ii,iii, ...
I	0,I,II,III, ...
a	0,a,b,...,z,aa,ab,...,zz,aaa, ...
A	0,A,B,...,Z,AA,AB,...,ZZ,AAA, ...

4. When defining many registers, it is necessary to remove registers after they are used. This will recapture internal storage space for newer registers. Registers are removed by entering the **.rr** request. The **.rr** request has the following form:

   .rr R

   where R is the number register name.

   Number register usage is explained in more detail in the "User Reference Manual".

FORMATTING YOUR DOCUMENT

1. Text Processors

Once you have created a file of text, you are ready to format it. Text processors prepare your files of text for printout on printers and phototypesetters. The DOCUMENTER'S WORKBENCH software provides the following text processors:

- **nroff** formats files for printing on typewriter-like devices (low-speed letter quality printers) and line printers.
- **otroff** formats files for a Wang Laboratories, Inc., C/A/T phototypesetter.
- **sroff** formats files for printing on typewriter-like devices and line printers.

- **troff** formats files for printing on a typesetter.

These text fonnatters are explained in detail in the "User Reference Manual".

The basic idea of formatting programs is that the text to be formatted contains within it "formatting commands" that determine in detail how the formatted text is to look. For example, there may be commands that specify how long lines are, whether to use single or double spacing, and the running titles to use on each page.

Formatting programs produce text with justified right margins, automatic page numbering and titling, automatic hyphenation, etc. The **nroff** (pronounced "en-roff") program is designed to produce output on terminals and line printers. The **nroff** program formats the text into a printable paginated document. The **troff** (pronounced "tee-roff") program is designed to drive a typesetter that produces high quality output on photographic paper. (This document was formatted with **troff**.) The **troff** output is a device-independent ASCII language that describes where characters are placed on a page by the typesetter. The output has been tailored to the resolution and font descriptions of a particular typesetter, but otherwise is independent of any particular device. The device-independent ASCII language is translated into the machine codes needed to run the particular typesetter by a program called a postprocessor.

2. Using Text Processors

The input form for invoking formatting programs is

nroff options files

or

otroff options files

or

sroff options files

or

troff options files

where options are optional arguments and files are the names of files containing the document to be formatted. For example, to produce a document in standard format using the memorandum macros, use the option "**−mm**" as follows:

troff −mm files ...

for the typesetter and

nroff −mm files ...

for a terminal.

The **−mm** argument tells **troff** and **nroff** to use the memorandum macro package of formatting requests. There are several similar packages. Check with a local expert to determine what is in common use on your machine.

3. Preprocessors

The text processing programs are powerful and flexible. In many ways these text processing programs resemble assembly programs. Many operations must be specified at a level of detail and in a form that is too difficult for most people to use effectively. This is why macro packages were created. Macro packages are easier to use than the detailed requests of the text processing programs. Preprocessors were created for the same reason. You would have difficulty producing text containing mathematics, tables, or simple pictures using text processing requests. The preprocessors will aid you in producing these special applications. The DOCUMENTER'S WORKBENCH software provides the following preprocessors:

- **eqn** converts files containing mathematical equations and expressions for **troff** or **otroff** output.
- **neqn** converts files containing mathematical equations and expressions for **nroff** output.
- **ocw** converts files of constant width text for **otroff** output.
- **pic** converts files containing simple pictures for **troff** output.
- **tbl** converts files containing tables for **nroff, otroff,** or **troff** output.

These preprocessors are explained in detail in the *Preprocessors Reference*—select code 307-153.

The mathematics, pictures, or tables can be interspersed in your text. Each of the preprocessors has special names to define the beginning and end of input for it as follows:

Preprocessor	Start Macro	End Macro
eqn	.EQ	.EN
neqn	.EQ	.EN
ocw	.CW	.CN
pic	.PS	.PE
tbl	.TS	.TE

Each preprocessor simply copies the input files to the standard output except for the lines between its start and end macros. These lines are assumed to describe a mathematical equation or expression, picture, or table. The preprocessor converts the lines between the start and end macros into text processing (**nroff**, **otroff**, and **troff**) requests.

4. Using Preprocessors

The general form for using a preprocessor is

preprocessor options files | textprocessor options

Some more specific examples are:

eqn options files | **troff** options

eqn **-T**cat options files | **otroff** options

neqn options files | **nroff** options

ocw options files | **otroff** options

pic options files | **troff** options

tbl options files | **nroff** options

See the manual entries in the Appendix to this book for more details.

5. Postprocessors

The DOCUMENTER'S WORKBENCH software provides the following post processors:

- **dx9700** prepares **troff** documents for the Xerox 9700 laser printer.
- **daps** prepares **troff** output for the AUTOLOGIC, Incorporated APS-5 typesetter.

- **di10** prepares **troff** output for the IMPRINT-10* laser printer.
- **otc** prepares **otroff** output for a Tektronix 4014.
- **tc** prepares **troff** output for a Tektronix 4014.
- **x9700** prepares **nroff** documents for the Xerox 9700 laser printer.

The general form for using a postprocessor follows:

preprocessor opt files | textprocessor opt | postprocessor opt

where "opt" is options.

Some examples of using a postprocessor follow:

tbl options files | **nroff** options | **x9700** options

troff –**T**aps options files | **daps** options

otroff options files | **otc** options

RECOMMENDATIONS AND SUMMARY

1. Inputting

Most documents go through several versions (always more than expected) before they are finally finished. Accordingly, you should do whatever possible to make revisions easy.

* IMPRINT is a registered trademark of the IMAGEN Corporation.

First, when you do the purely mechanical operation of typing, type so that later editing will be easy. Start each sentence on a new line. Make lines short, and break lines at natural places, such as after commas and semicolons, rather than randomly. Since most people change documents by rewriting phrases and adding, deleting, and rearranging sentences, these precautions simplify any editing needed later.

Keep the individual files of a document down to modest size, perhaps less than 20,000 charcters. Larger files edit more slowly. If a mistake is made, it is better to clobber a small file than a big one. Split the files at natural boundaries in the document for the same reasons that you start each sentence on a new line.

2. Formatting

The second aspect of allowing documents to be easily changed is to not specify the formatting details too early. One advantage of formatting packages is permitting format decisions to be delayed until the last possible moment. Indeed, until a document is printed, it is not even decided whether it will be typeset or printed out on a line printer.

As a rule of thumb, a document should be produced by a set of requests or commands (macros) for all but the most trivial jobs. The macros used are defined either by using an existing macro package (the recommended way) or by defining your own **nroff** and **troff** macros. As long as the text is entered in some systematic way, it can always be cleaned up and formatted by a judicious combination of editing commands and macro definitions.

3. Printing

If you are a beginning user, obtain a hard copy of documents more than a few pages long for making corrections. Beginning users tend to make more mistakes in large documents than they can remember. Making major corrections at your desk prevents terminal tie-up. Mark the copy with corrections. Refer to this manual for DOCUMENTER'S WORKBENCH software and to the editing guides in the Runtime System manual when necessary. Then enter the corrections at the terminal in the unformatted raw text files, not the formatted files.

The next step is to determine if the corrections worked as follows:

- If there were many corrections, reformat and print the entire document. Check the output to ensure all changes worked.
- For minor changes, reformat the corrected pages only. This can save time. For example, assume your document is 60 pages and corrections were made on pages 16, 23, 47, and 58. These pages can be formatted and stored in a file as follows:

  nroff -o16,23,47,58 files>FILE

  The same example that uses a preprocessor and postprocessor would be:

  tbl files | **nroff** -o16,23,47,58 >FILE

  The formatted file "FILE" can then be edited or printed to determine if the changes worked. Assume your printer is connected to your terminal and prints one page in two minutes. Instead of waiting two hours for a 60-page document to print out, the wait is about eight minutes. This saves paper and time, and allows others to use the terminal.

  If you are an experienced user, view your formatted text on the screen. Text files formatted by **nroff** and **sroff** are printed on your terminal screen by default. Remember the mistakes and correct the input using the text editor of your choice. Follow the same advice given to beginners for determining if the corrections worked.

Chapter 2

USER REFERENCE MANUAL

The following pages contain descriptions of the commands that are part of the DOCUMENTER'S WORKBENCH software. It is intended that these pages serve as a memory jogger for more experienced users and a source of information for less experienced users.

(PDP-11 only)

NAME

cat – phototypesetter interface

DESCRIPTION

Cat provides the interface to a Wang Laboratories, Inc. C/A/T phototypesetter. Bytes written on the file specify font, size, and other control information as well as the characters to be flashed. The coding will not be described here.

Only one process may have this file open at a time. It is write-only.

FILES

/dev/cat

SEE ALSO

nroff(1).

Wang Laboratories, Inc. specification (available on request).

NAME

daps, di10 − Postprocessors for the Autologic APS-5 phototypesetter and the Imagen Imprint-10 laser printer

SYNOPSIS

daps [option] ... [file] ...
di10 [option] ... [file] ...

DESCRIPTION

Daps and *di10* (formerly known as *dcan*) print *files* created by *troff*(1) on an Autologic APS-5 phototypesetter or on an Imagen Imprint-10 laser printer. If no *file* is mentioned, the standard input is printed. The following options are understood.

−b Report whether the typesetter is busy; do not print.

−h*string*
Print *string* in this job's header. A header will only be generated if either this option or the -H option is used. (*daps* only)

−H*file* Print the first line from *file* in this job's header. (*daps* only)

−o*list* Print pages whose numbers are given in the comma-separated *list*. The list contains single numbers *N* and ranges *N1* **−***N2*. A missing *N1* means the lowest-numbered page, a missing *N2* means the highest.

−r Report the number of 11-inch pages generated by this job. (*daps* only)

−s*n* Stop after every *n* pages of output. Continue when the PROCEED button is pushed on the typesetter.

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

−w Wait for typesetter to become free, then print.

The *files* submitted to *daps* should be prepared under the **−Taps** option of *troff*. *Di10* is a phototypesetter simulator and can handle *troff* output prepared for any supported typesetter. However, files sent to *di10* will look best when prepared with the **-Ti10** option of *troff*.

FILES

/dev/aps	APS-5 phototypesetter device
/usr/lib/font/devaps/*	description files for APS-5
/usr/lib/font/devi10/*	description files for Imagen Imprint-10
/usr/lib/font/devi10/rasti10/*	raster files for Imprint-10
/tmp/dcan*	output of *di10* ready for Imagen

SEE ALSO

tc(1), troff(1), troff(5).

BUGS

Installations with an Autologic APS-5 phototypesetter should be aware that getting a good match to their Autologic fonts will almost certainly require hand-tuning of the distributed font description files.

NAME

dx9700 – prepare troff documents for the Xerox 9700 printer

SYNOPSIS

dx9700 *name*

DESCRIPTION

The *dx9700* filter is a post-processor for device independent *troff* output, and produces codes suitable for being sent to a Xerox 9700 laser printer.

The single argument to *dx9700* should be the *name* part of the **–T***name* argument given to *troff*.

The output of the *dx9700* filter should be directed to the input of a Xerox 9700 printer.

Note that the Xerox 9700 treats different point sizes as different fonts. Hence, the font tables specified to *troff*(1) and *dx9700* actually specify a family of typefaces and point sizes. The font families that are supported for the Xerox 9700 and that can be specified to *troff* using the **-T** option follow:

name	**contains**
X97.tim10p	Times, 7, 10, and 15 point
X97.tim12p	Times, 9, 12, and 17 point

SEE ALSO

troff(1), troff(5).

BUGS

Special fonts for the Xerox 9700 printer are needed to use with this post-processor.

NAME

eqn, neqn, checkeq − format mathematical text for nroff or troff

SYNOPSIS

eqn [−dxy] [−pn] [−sn] [−fn] [−Tdest] [files]

neqn [−dxy] [−pn] [−sn] [−fn] [files]

checkeq [files]

DESCRIPTION

Eqn is a *troff*(1) preprocessor for typesetting mathematical text on a phototypesetter, while *neqn* is used for the same purpose with *nroff* on typewriter-like terminals. Usage is almost always:

```
eqn files | troff
neqn files | nroff
```

or equivalent. If no files are specified (or if − is specified as the last argument), these programs read the standard input. *Eqn* prepares output for the typesetter named in the **−T** option. Currently supported devices are **−Taps** (Autologic APS-5), **-TX97** (Xerox 9700), **-Ti10** (Imagen Imprint-10), and **−Tcat** (Wang CAT). Default is **−Taps**.

A line beginning with **.EQ** marks the start of an equation; the end of an equation is marked by a line beginning with **.EN**. Neither of these lines is altered, so they may be defined in macro packages to get centering, numbering, etc. It is also possible to designate two characters as *delimiters*; subsequent text between delimiters is then treated as *eqn* input. Delimiters may be set to characters *x* and *y* with the command-line argument **−d***xy* or (more commonly) with **delim** *xy* between **.EQ** and **.EN**. The left and right delimiters may be the same character; the dollar sign is often used as such a delimiter. Delimiters are turned off by **delim off**. All text that is neither between delimiters nor between **.EQ** and **.EN** is passed through untouched.

The program *checkeq* reports missing or unbalanced delimiters and **.EQ**/**.EN** pairs.

Tokens within *eqn* are separated by spaces, tabs, new-lines, braces, double quotes, tildes, and circumflexes. **Braces** {} are used for grouping; generally speaking, anywhere a single character such as *x* could appear, a complicated construction enclosed in braces may be used instead. Tilde (~) represents a full space in the output, circumflex (^) half as much.

Subscripts and superscripts are produced with the keywords **sub** and **sup**. Thus *x sub j* makes x_j, *a sub k sup 2* produces a_k^2, while $e^{x^2+y^2}$ is made with *e sup {x sup 2 + y sup 2}*. Fractions are made with **over**: *a over b* yields $\frac{a}{b}$; **sqrt** makes square roots: *1 over sqrt {ax sup 2+bx+c}* results in $\frac{1}{\sqrt{ax^2+bx+c}}$.

The keywords **from** and **to** introduce lower and upper limits: $\lim_{n \to \infty} \sum_0^n x_i$ is made with *lim from {n −> inf} sum from 0 to n x sub i*. Left and right brackets, braces, etc., of the right height are made with **left** and **right**: *left [x sup 2 + y sup 2 over alpha right] ~=~ 1* produces $\left[x^2+\frac{y^2}{\alpha}\right] = 1$.

Legal characters after **left** and **right** are braces, brackets, bars, **c** and **f** for ceiling and floor, and "" for nothing at all (useful for a right-side-only bracket). A **left** *thing* need not have a matching **right** *thing*.

Vertical piles of things are made with **pile**, **lpile**, **cpile**, and **rpile**: *pile {a above b above c}* produces $\begin{matrix} a \\ b \\ c \end{matrix}$. Piles may have arbitrary numbers of elements; **lpile** left-justifies, **pile** and **cpile** center (but with different vertical spacing), and **rpile** right justifies. Matrices are made with **matrix**: *matrix { lcol { x sub i above y sub 2 } ccol { 1 above 2 } }* produces $\begin{matrix} x_i & 1 \\ y_2 & 2 \end{matrix}$. In addition, there is **rcol** for a right-justified column.

Diacritical marks are made with **dot**, **dotdot**, **hat**, **tilde**, **bar**, **vec**, **dyad**, and **under**: *x dot = f(t) bar* is $\dot{x} = \overline{f(t)}$, *y dotdot bar ~=~ n under* is $\ddot{y} \sim = \sim \underline{n}$, and *x vec ~=~ y dyad* is $\vec{x} = \vec{\vec{y}}$.

Point sizes and fonts can be changed with **size** *n* or **size** ±*n*, **roman**, **italic**, **bold**, and **font** *n*. Point sizes and fonts can be changed globally in a document by **gsize** *n* and **gfont** *n*, or by the command-line arguments **−s***n* and **−f***n*.

Normally, subscripts and superscripts are reduced by 3 points from the previous size; this may be changed by the command-line argument **−p***n*.

Successive display arguments can be lined up. Place **mark** before the desired lineup point in the first equation; place **lineup** at the place that is to line up vertically in subsequent equations.

Shorthands may be defined or existing keywords redefined with **define**:

define thing % replacement %

defines a new token called *thing* that will be replaced by *replacement* whenever it appears thereafter. The % may be any character that does not occur in *replacement*.

Keywords such as **sum** (∑), **int** (∫), **inf** (∞), and shorthands such as **>=** (⩾), **!=** (≠), and **−>** (→) are recognized. Greek letters are spelled out in the desired case, as in **alpha** (α), or GAMMA (Γ). Mathematical words such as **sin**, **cos**, and **log** are made Roman automatically. *Troff*(1) four-character escapes such as **\(dd** (‡) and **\(sc** (§) may be used anywhere. Strings enclosed in double quotes ("...") are passed through untouched; this permits keywords to be entered as text, and can be used to communicate with *troff*(1) when all else fails. Full details are given in the manual cited below.

SEE ALSO

mm(1), mmt(1), nroff(1), tbl(1), troff(1), eqnchar(5), mm(5), mv(5).

DOCUMENTER'S WORKBENCH Software Preprocessor Reference.

BUGS

To embolden digits, parentheses, etc., it is necessary to quote them, as in **bold "12.3"**.

See also *BUGS* under *troff*(1).

NAME

eqnchar – special character definitions for eqn and neqn

SYNOPSIS

eqn /usr/pub/eqnchar [files] | **troff** [options]

neqn /usr/pub/eqnchar [files] | **nroff** [options]

eqn -Taps /usr/pub/apseqnchar [files] | **troff** [options]

eqn -Tcat /usr/pub/cateqnchar [files] | **otroff** [options]

DESCRIPTION

Eqnchar contains *troff*(1) and *nroff*(1) character definitions for constructing characters that are not available on a phototypesetter. These definitions are primarily intended for use with *eqn*(1) and *neqn*; *eqnchar* contains definitions for the following characters:

ciplus	*ciplus*	||	||	*square*	*square*
citimes	*citimes*	*langle*	*langle*	*circle*	*circle*
wig	*wig*	*rangle*	*rangle*	*blot*	*blot*
−wig	*−wig*	*hbar*	*hbar*	*bullet*	*bullet*
>wig	*>wig*	*ppd*	*ppd*	*prop*	*prop*
<wig	*<wig*	*<−>*	*<→*	*empty*	*empty*
=wig	*=wig*	*<=>*	*≤>*	*member*	*member*
star	*star*	*|<*	*|<*	*nomem*	*nomem*
bigstar	*bigstar*	*|>*	*|>*	*cup*	*cup*
=dot	*=dot*	*ang*	*ang*	*cap*	*cap*
orsign	*orsign*	*rang*	*rang*	*incl*	*incl*
andsign	*andsign*	*3dot*	*3dot*	*subset*	*subset*
=del	*=del*	*thf*	*thf*	*supset*	*supset*
oppA	*oppA*	*quarter*	*quarter*	*!subset*	*!subset*
oppE	*oppE*	*3quarter*	*3quarter*	*!supset*	*!supset*
angstrom	*angstrom*	*degree*	*degree*	*scrL*	*scrL*
==<	==<	==>	==>		

Apseqnchar is a version of *eqnchar* tailored for the Autologic APS-5 phototypesetter. This will not look optimal on other phototypesetters. Similarly, *cateqnchar* is the old *eqnchar* tailored for the Wang CAT and the old *otroff*. Until a phototypesetter-independent version of *eqnchar* is available, *eqnchar* should be a link to the default version on each system. The standard default is *apseqnchar*.

FILES

/usr/pub/eqnchar
/usr/pub/apseqnchar
/usr/pub/cateqnchar

SEE ALSO

eqn(1), nroff(1), troff(1).

NAME

font – description files for device-independent troff

SYNOPSIS

troff **–T**ptty ...

DESCRIPTION

For each phototypesetter supported by *troff*(1) and available on this system, there is a directory containing files describing the device and its fonts. This directory is named **/usr/lib/font/dev***ptty* where *ptty* is the name of the phototypesetter. Currently the only *ptty* supported is aps for the Autologic APS–5.

For a particular phototypesetter, *ptty*, the ASCII file *DESC* in the directory **/usr/lib/font/dev***ptty* describes its characteristics. Each line starts with a word identifying the characteristic and followed by appropriate specifiers. Blank lines and lines beginning with a # are ignored.

The legal lines for *DESC* are:

res *num*	resolution of device in basic increments per inch
hor *num*	smallest unit of horizontal motion
vert *num*	smallest unit of vertical motion
unitwidth *num*	pointsize in which widths are specified
sizescale *num*	scaling for fractional pointsizes
paperwidth *num*	width of paper in basic increments
paperlength *num*	length of paper in basic increments
spare1 *num*	available for use
spare2 *num*	available for use
sizes *num num* ...	list of pointsizes available on typesetter
fonts *num name* ...	number of initial fonts followed by the names of the fonts. For example: fonts 4 R I B S
charset	this always comes last in the file and is on a line by itself. Following it is the list of special character names for this device. Names are separated by a space or a newline. The list can be as long as necessary. Names not in this list are not allowed in the font description files.

Res is the basic resolution of the device in increments per inch. **Hor** and **vert** describe the relationships between motions in the horizontal and vertical directions. If the device is capable of moving in single basic increments in both directions, both **hor** and **vert** would have values of 1. If the vertical motions only take place in multiples of two basic units while the horizontal motions take place in the basic increments, then **hor** would be 1, while **vert** would be 2. **Unitwidth** is the pointsize in which all width tables in the font description files are given. *Troff* automatically scales the widths from the **unitwidth** size to the pointsize it is working with. **Sizescale** is not currently used and is 1. **Paperwidth** is the width of the paper in basic increments. The APS-5 is 6120 increments wide. **Paperlength** is the length of a sheet of paper in the basic increments.

For each font supported by the phototypesetter, there is also an ASCII file with the same name as the font (e.g., **R**, **I**, **CW**). The format for a font description file is:

name *name*	name of the font, such as **R** or **CW**
internalname *name*	internal name of font
special	sets flag indicating that the font is special
ligatures *name* ... **0**	Sets flag indicating font has ligatures. The list of ligatures follows and is terminated by a zero. Accepted ligatures are: **ff fi fl ffi ffl**.
spare1	available for use
spacewidth *num*	width of space if something other than 1/3 of \(em is desired as a space.
charset	The charset must come at the end. Each line following the word *charset* describes one character in the font. Each line has one of two formats: *name width kerning code* *name* "

where *name* is either a single ASCII character or a special character name from the list found in *DESC*. The width is in basic increments. The kerning information is 1 if the character descends below the line, 2 if it rises above the letter 'a', and 3 if it both rises and descends. The kerning information for special characters is not used and so may be 0. The code is the number sent to the typesetter to produce the character. The second format is used to indicate that the character has more than one name. The double quote indicates that this name has the same values as the preceding line. The kerning and code fields are not used if the width field is a double quote character.

Troff and its postprocessors read this information from binary files produced from the ASCII files by a program distributed with *troff* called *makedev*. For those with a need to know, a description of the format of these files follows:

The file *DESC.out* starts with the *dev* structure, defined by *dev.h*:

```
 /*
dev.h: characteristics of a typesetter
* /

struct dev {
short     filesize;      /* number of bytes in file, */
                /* excluding dev part */
short     res;           /* basic resolution in goobies/inch */
short     hor;           /* goobies horizontally */
short     vert;
short     unitwidth;     /* size at which widths are given*/
short     nfonts;        /* number fonts physically available */
short     nsizes;        /* number of pointsizes */
short     sizescale;     /* scaling for fractional pointsizes */
short     paperwidth;    /* max line length in units */
short     paperlength;    /* max paper length in units */
short     nchtab;        /* number of funny names in chtab */
short     lchname;       /* length of chname table */
short     spare1;        /* in case of expansion */
short     spare2;
};
```

Filesize is just the size of everything in *DESC.out* excluding the *dev* structure. *Nfonts* is the number of different font positions available. *Nsizes* is the number of different pointsizes supported by this typesetter. *Nchtab* is the number of special character names. *Lchname* is the total number of characters, including nulls, needed to list all the special character names. At the end of the structure are two spares for later expansions.

Immediately following the *dev* structure are a number of tables. First is the *sizes* table, which contains *nsizes* + 1 shorts(a null at the end), describing the pointsizes of text available on this device. The second table is the *funny_char_index_table*. It contains indices into the table which follows it, the *funny_char_strings*. The indices point to the beginning of each special character name which is stored in the *funny_char_strings* table. The *funny_char_strings* table is *lchname* characters long, while the *funny_char_index_table* is *nchtab* shorts long.

Following the *dev* structure will occur *nfonts* {*font*}*.out* files, which are used to initialize the font positions. These {*font*}*.out* files, which also exist as separate files, begin with a *font* structure and then are followed by four character arrays:

```
struct font {      /* characteristics of a font */
char nwfont;      /* number of width entries */
char specfont;    /* 1 == special font */
char  ligfont;    /* 1 == ligatures exist on this font */
char spare1;      /* unused for now */
char namefont[10]; /* name of this font, e.g., R */
char intname[10];  /* internal name of font, in ASCII */
};
```

The *font* structure tells how many defined characters there are in the font, whether the font is a "special" font and if it contains ligatures. It also has the ASCII name of the font, which should match the name of the file it appears in, and the internal name of the font on the typesetting device (*intname*). The internal name is independent of the font position and name that *troff* knows about. For example, you might say mount R in position 4, but when asking the typesetter to actually produce a character from the R font, the postprocessor which instructs the typesetter would use *intname*.

The first three character arrays are specific for the font and run in parallel. The first array, *widths*, contains the width of each character relative to *unitwidth*. *Unitwidth* is defined in *DESC*. The second array, *kerning*, contains kerning information. If a character rises above the letter 'a', 02 is set. If it descends below the line, 01 is set. The third array, *codes*, contains the code that is sent to the typesetter to produce the character.

The fourth array is defined by the device description in *DESC*. It is the *font_index_table*. This table contains indices into the *width*, *kerning*, and *code* tables for each character. The order that characters appear in these three tables is arbitrary and changes from one font to the next. In order for *troff* to be able to translate from ASCII and the special character names to these arbitrary tables, the *font_index_table* is created with an order which is constant for each device. The number of entries in this table is 96 plus the number of special character names for this device. The value 96 is 128 - 32, the number of printable characters in the ASCII alphabet. To determine whether a

normal ASCII character exists, *troff* takes the ASCII value of the character, subtracts 32, and looks in the *font_index_table*. If it finds a 0, the character is not defined in this font. If it finds anything else, that is the index into *widths*, *kerning*, and *codes* that describe that character.

To look up a special character name, for example **\(pl**, the mathematical plus sign, and determine whether it appears in a particular font or not, the following procedure is followed. A *counter* is set to 0 and an index to a special character name is picked out of the *counter'th* position in the *funny_char_index_table*. A string comparision is performed between *funny_char_strings [funny_char_index_table [counter]]* and the special character name, in our example **pl**, and if it matches, then *troff* refers to this character as (96 + *counter*). When it wants to determine whether a specific font supports this character, it looks in *font_index_table[(96+counter)]*, (see below), to see whether there is a 0, meaning the character does not appear in this font, or number, which is the index into the *widths*, *kerning*, and *codes* tables.

Notice that since a value of 0 in the *font_index_table* indicates that a character does not exist, the 0th element of the *width*, *kerning*, and *codes* arrays are not used. For this reason the 0th element of the *width* array can be used for a special purpose, defining the width of a space for a font. Normally a space is defined by *troff* to be 1/3 of the width of the \(em character, but if the 0th element of the *width* array is non-zero, then that value is used for the width of a space.

SEE ALSO

troff(1), troff(5).

FILES

/usr/lib/font/dev{X}/DESC.out description file for phototypesetter X
/usr/lib/font/dev{X}/{font}.out font description files for phototypesetter X

NAME

macref – produce cross-reference listing of macro files

SYNOPSIS

macref [**–t**] [**–s**] [**–n**] file ...

DESCRIPTION

The *macref* program reads the named files (which are assumed to be *nroff*(1)/*troff*(1) input) and produces a cross-reference listing of the symbols in the input.

A **–t** in the command line causes a macro table of contents to be printed. A **–s** causes symbol use statistics to be output.

The default output is a list of the symbols found in the input, each accompanied by a list of all references to that symbol. (This output may be defeated by using a **–n** in the command line). The symbols are listed alphabetically in the leftmost column, with the references following to the right. Each reference is given in the form:

[[(*NMname*)] *Mname*–] *type lnum* [#]

where the fields have the following meanings:

Mname the name of the macro within which the reference occurs. This field is missing if the reference occurs at the text level. Any names listed in the *NMname* part are macros within which *Mname* is defined.

type the type associated, by context, with this occurrence of the symbol. The types may be:

- **r** request
- **m** macro
- **d** diversion
- **s** string
- **n** number register
- **p** parameter (e.g.,\$x is a parameter reference to x. Note that parameters are never modified, and that the only valid parameter symbol names are 1, 2, ... 9).

lnum the line number on which the reference occurred.

\# this reference modifies the value of the symbol.

Generated names are listed under the artificial symbol name "~sym".

SEE ALSO

nroff(1), troff(1).

NAME

man, manprog − print entries in this manual

SYNOPSIS

man [options] [section] titles

/usr/lib/manprog file

DESCRIPTION

Man locates and prints the entry of this manual named *title* in the specified *section*. (For historical reasons, the word "page" is often used as a synonym for "entry" in this context.) The *title* is entered in lower case. The *section* number may not have a letter suffix. If no *section* is specified, the whole manual is searched for *title* and all occurrences of it are printed. *Options* and their meanings are:

−t	Typeset the entry in the default format (8.5"×11").
−s	Typeset the entry in the small format (6"×9").
−Tcat	Use *otroff*(1) to generate output for an on-line Wang CAT phototypesetter.
−D4014	Display the typeset output on a TEKTRONIX 4014 terminal using *tc*(1).
−Dtek	Same as **−D4014**.
−Di10	Send typeset output to the local Imagen Imprint-10 laser printer.
−T*term*	If *term* is one of the recognized *troff* devices (see *troff*(1)), format the entry for that device. Otherwise format the entry using *nroff* and print it on the standard output (usually, the terminal); *term* is the terminal type (see *term*(5) and the explanation below); for a list of recognized values of *term*, type **help term2**. The default value of *term* is **450**.
−w	Print on the standard output only the *pathnames* of the entries, relative to **/usr/man**, or to the current directory for −d option.
−d	Search the current directory rather than **/usr/man**; requires the full file name (e.g., **cu.1c**, rather than just **cu**).
−12	Indicates that the manual entry is to be produced in 12-pitch. May be used when **$TERM** (see below) is set to one of **300, 300s, 450**, and **1620**. (The pitch switch on the DASI 300 and 300s terminals must be manually set to **12** if this option is used.)
−c	Causes *man* to invoke *col*(1); note that *col*(1) is invoked automatically by *man* unless *term* is one of **300, 300s, 450, 37, 4000a, 382, 4014, tek, 1620**, and **X**.
−y	Causes *man* to use the non-compacted version of the macros.
−z	Invokes no output filter to process or redirect the output of *troff*(1).

The above *options* other than **−d, −c**, and −y are mutually exclusive, except that the **−s** and **−z** options may be used in conjunction with any typesetter option (6"×9" pages may be produced with *nroff* by including the **−rs1** option). Any other *options* are passed to *troff*, *nroff*, or the *man*(5) macro package.

When using *nroff*, *man* examines the environment variable **$TERM** (see *environ*(5)) and attempts to select options to *nroff*, as well as filters, that adapt the output to the terminal being used. The **−T***term* option overrides the value of **$TERM**; in particular, one should use **−Tlp** when sending the output of *man* to a line printer.

Section may be changed before each *title*.

As an example:

```
man man
```

would reproduce on the terminal this entry, as well as any other entries named *man* that may exist in other sections of the manual, e.g., *man*(5).

If the first line of the input for an entry consists solely of the string:

```
'\" x
```

where *x* is any combination of the two characters **e**, and **t**, and where there is exactly one blank between the double quote (") and *x*, then *man* will preprocess its input through the appropriate combination of *eqn*(1) (*neqn* for *nroff*) and *tbl*(1), respectively. If *eqn* or *neqn* are invoked, they will automatically read the file **/usr/pub/eqnchar** (see *eqnchar*(5)).

The *man* command executes *manprog* that takes a file name as its argument. *Manprog* calculates and returns a string of three register definitions used by the formatters identifying the date the file was last modified. The returned string has the form:

−rd*day* **−rm***month* **−ry***year*

and is passed to *nroff* which sets this string as variables for the *man* macro package. Months are given from 0 to 11, therefore month is always 1 less than the actual month. The *man* macros calculate the correct month. If the *man* macro package is invoked as an option to *nroff*/*troff* (i.e., *nroff −man file*), then the current day/month/year is used as the printed date.

FILES

/usr/man/u_man/man[1,6]/*	the *UNIX System User Reference Manual*
/usr/man/a_man/man[1,7,8]/*	the *UNIX System Administrator Reference Manual*
/usr/man/p_man/man[2-5]/*	the *UNIX System Programmer Reference Manual*
/usr/man/local/man[1-8]/*	local additions
/usr/man/*/man[1-8]/*	any other additions
/usr/lib/manprog	calculates modification dates of entries

SEE ALSO

daps(1), eqn(1), nroff(1), tbl(1), tc(1), troff(1), environ(5), man(5), term(5).

BUGS

All entries are supposed to be reproducible either on a typesetter or on a terminal. However, on a terminal some information is necessarily lost.

Pages bearing the same name in all three manuals will result in the *UNIX System Administrator Reference Manual* entry being printed first, if no *section* argument is supplied.

6"×9" manual entries formatted by *nroff* (with the **−rs1** option) are not guaranteed to look as good as regular-sized entries.

NAME

man − macros for formatting entries in this manual

SYNOPSIS

nroff −man files

troff −man [−rs1] files

DESCRIPTION

These *troff*(1) macros are used to lay out the format of the entries of this manual. A skeleton entry may be found in the file **/usr/man/u_man/man0/skeleton**. These macros are used by the *man*(1) command.

The default page size is 8.5"×11", with a 6.5"×10" text area; the **−rs1** option reduces these dimensions to 6"×9" and 4.75"×8.375", respectively; this option (which is *not* effective in *nroff*(1)*)* also reduces the default type size from 10-point to 9-point, and the vertical line spacing from 12-point to 10-point. The **−rV2** option may be used to set certain parameters to values appropriate for certain Versatec printers: it sets the line length to 82 characters, the page length to 84 lines, and it inhibits underlining; this option should not be confused with the **−Tvp** option of the *man*(1) command, which is available at some UNIX system sites.

Any *text* argument below may be one to six "words". Double quotes (**""**) may be used to include blanks in a "word". If *text* is empty, the special treatment is applied to the next line that contains text to be printed. For example, **.I** may be used to italicize a whole line, or **.SM** followed by **.B** to make small bold text. By default, hyphenation is turned off for *nroff*(1), but remains on for *troff*(1).

Type font and size are reset to default values before each paragraph and after processing font- and size-setting macros, e.g., **.I, .RB, .SM**. Tab stops are neither used nor set by any macro except **.DT** and **.TH**.

Default units for indents *in* are ens. When *in* is omitted, the previous indent is used. This remembered indent is set to its default value (7.2 ens in *troff*(1), 5 ens in *nroff* this corresponds to 0.5" in the default page size) by **.TH, .P**, and **.RS**, and restored by **.RE**.

.TH *t s c n* Set the title and entry heading; *t* is the title, *s* is the section number, *c* is extra commentary, e.g., "local", *n* is new manual name. Invokes **.DT** (see below).

.SH *text* Place subhead *text*, e.g., **SYNOPSIS**, here.

.SS *text* Place sub-subhead *text*, e.g., **Options**, here.

.B *text* Make *text* bold.

.I *text* Make *text* italic.

.SM *text* Make *text* 1 point smaller than default point size.

.RI *a b* Concatenate roman *a* with italic *b*, and alternate these two fonts for up to six arguments. Similar macros alternate between any two of roman, italic, and bold:

.IR .RB .BR .IB .BI

.P Begin a paragraph with normal font, point size, and indent. **.PP** is a synonym for **.P**.

.HP *in* Begin paragraph with hanging indent.

.TP *in* Begin indented paragraph with hanging tag. The next line that contains text to be printed is taken as the tag. If the tag does not fit, it is printed on a separate line.

.IP *t in* Same as **.TP** *in* with tag *t*; often used to get an indented paragraph without a tag.

.RS *in* Increase relative indent (initially zero). Indent all output an extra *in* units from the current left margin.

.RE *k* Return to the *k*th relative indent level (initially, *k*=1; *k*=0 is equivalent to *k*=1); if *k* is omitted, return to the most recent lower indent level.

.PM *m* Produces proprietary markings; where *m* may be **P** for **PRIVATE**, **N** for **NOTICE**, **BP** for **BELL LABORATORIES PROPRIETARY**, or **BR** for **BELL LABORATORIES RESTRICTED**.

.DT Restore default tab settings (every 7.2 ens in *troff*(1), 5 ens in *nroff*(1)).

.PD *v* Set the interparagraph distance to *v* vertical spaces. If *v* is omitted, set the interparagraph distance to the default value (0.4v in *troff*(1), 1v in *nroff*(1)).

The following *strings* are defined:

***R** ® in *troff*(1), **(Reg.)** in *nroff*.
***S** Change to default type size.
***(Tm** Trademark indicator.

The following *number registers* are given default values by **.TH**:

IN Left margin indent relative to subheads (default is 7.2 ens in *troff*(1), 5 ens in *nroff*(1)).
LL Line length including **IN**.
PD Current interparagraph distance.

CAVEATS

In addition to the macros, strings, and number registers mentioned above, there are defined a number of *internal* macros, strings, and number registers. Except for names predefined by *troff (1)* and number registers **d**, **m**, and **y**, all such internal names are of the form *XA*, where *X* is one of),], and }, and *A* stands for any alphanumeric character.

If a manual entry needs to be preprocessed by *eqn*(1) (or *neqn*), and/or *tbl*(1), it must begin with a special line (described in *man*(1)), causing the *man* command to invoke the appropriate preprocessor(s).

The programs that prepare the Table of Contents and the Permuted Index for this Manual assume the *NAME* section of each entry consists of a single line of input that has the following format:

name[, name, name ...] \- explanatory text

The macro package increases the inter-word spaces (to eliminate ambiguity) in the *SYNOPSIS* section of each entry.

The macro package itself uses only the roman font (so that one can replace, for example, the bold font by the constant-width font (CW). Of course, if the input text of an entry contains requests for other fonts (e.g., **.I**, **.RB**, **\fI**), the corresponding fonts must be mounted.

FILES

/usr/lib/tmac/tmac.an
/usr/lib/macros/cmp.n.[dt].an
/usr/lib/macros/ucmp.n.an
/usr/man/[uap]_man/man0/skeleton

SEE ALSO

ocw(1), eqn(1), man(1), nroff(1), tbl(1), tc(1), troff(1).

BUGS

If the argument to .TH contains *any* blanks and is *not* enclosed by double quotes (""), there will be strange irregular dots on the output.

NAME

mm, osdd, checkmm − print/check documents formatted with the MM macros

SYNOPSIS

mm [options] [files]

osdd [options] [files]

checkmm [files]

DESCRIPTION

Mm can be used to type out documents using *nroff* and the MM text-formatting macro package. It has options to specify preprocessing by *tbl*(1) and/or *neqn* (see *eqn*(1)) and postprocessing by various terminal-oriented output filters. The proper pipelines and the required arguments and flags for *nroff* and MM are generated, depending on the options selected.

Osdd is equivalent to the command **mm −mosd**. For more information about the OSDD adapter macro package, see *mosd*(5).

Options for *mm* are given below. Any other arguments or flags (e.g., **−rC3**) are passed to *nroff* or to MM, as appropriate. Such options can occur in any order, but they must appear before the *files* arguments. If no arguments are given, *mm* prints a list of its options.

−T*term* Specifies the type of output terminal; for a list of recognized values for *term*, type **help term2**. If this option is *not* used, *mm* will use the value of the shell variable **$TERM** from the environment (see *profile*(4) and *environ*(5)) as the value of *term*, if **$TERM** is set; otherwise, *mm* will use **450** as the value of *term*. If several terminal types are specified, the last one takes precedence.

−12 Indicates that the document is to be produced in 12-pitch. May be used when **$TERM** is set to one of **300, 300s, 450,** and **1620.** (The pitch switch on the DASI 300 and 300s terminals must be manually set to **12** if this option is used.)

−c Causes *mm* to invoke *col*(1); note that *col*(1) is invoked automatically by *mm* unless *term* is one of **300, 300s, 450, 37, 4000a, 382, 4014, tek, 1620,** and **X.**

−e Causes *mm* to invoke *neqn*; also causes *neqn* to read the **/usr/pub/eqnchar** file (see *eqnchar*(5)).

−t Causes *mm* to invoke *tbl*(1).

−E Invokes the **−e** option of *nroff*.

−y Causes *mm* to use the non-compacted version of the macros (see *mm*(5)).

As an example (assuming that the shell variable **$TERM** is set in the environment to **450**), the two command lines below are equivalent:

```
mm −t −rC3 −12 ghh*
tbl ghh* | nroff −cm −T450−12 −h −rC3
```

Mm reads the standard input when **−** is specified instead of any file names. (Mentioning other files together with **−** leads to disaster.) This option allows *mm* to be used as a filter, e.g.:

```
cat dws | mm −
```

Checkmm is a program for checking the contents of the named *files* for errors in the use of the Memorandum Macros, missing or unbalanced *neqn* delimiters, and **.EQ/.EN** pairs. Note: The user need not use the *checkeq* program (see *eqn*(1)). Appropriate messages are produced. The program skips all directories, and if no file name is given, standard input is read.

HINTS

1. *Mm* invokes *nroff* with the **−h** flag. With this flag, *nroff* assumes that the terminal has tabs set every 8 character positions.
2. Use the **−o***list* option of *nroff* to specify ranges of pages to be output. Note, however, that *mm*, if invoked with one or more of the **−e**, **−t**, and **−** options, *together* with the **−o***list* option of *nroff* may cause a harmless "broken pipe" diagnostic if the last page of the document is not specified in *list*.
3. If you use the **−s** option of *nroff* (to stop between pages of output), use line-feed (rather than return or new-line) to restart the output. The **−s** option of *nroff* does not work with the **−c** option of *mm*, or if *mm* automatically invokes *col*(1) (see **−c** option above).
4. If you lie to *mm* about the kind of terminal its output will be printed on, you will get (often subtle) garbage; however, if you are redirecting output into a file, use the **−T37** option, and then use the appropriate terminal filter when you actually print that file.

SEE ALSO

col(1), env(1), eqn(1), greek(1), mmt(1), nroff(1), tbl(1), profile(4), mm(5), mosd(5), term(5).

DIAGNOSTICS

mm "mm: no input file" if none of the arguments is a readable file and *mm* is not used as a filter.

checkmm "Cannot open *filename*" if file(s) is unreadable. The remaining output of the program is diagnostic of the source file.

NAME

mm − the MM macro package for formatting documents

SYNOPSIS

mm [options] [files]

nroff −mm [options] [files]

nroff −cm [options] [files]

mmt [options] [files]

troff −mm [options] [files]

DESCRIPTION

This package provides a formatting capability for a very wide variety of documents. It is the standard package used by the BTL typing pools and documentation centers. The manner in which a document is typed in and edited is essentially independent of whether the document is to be eventually formatted at a terminal or is to be phototypeset. See the references below for further details.

The **−mm** option causes *nroff*(1) and *troff*(1) to use the non-compacted version of the macro package, while the **−cm** option results in the use of the compacted version, thus speeding up the process of loading the macro package.

FILES

/usr/lib/tmac/tmac.m	pointer to the non-compacted version of the package
/usr/lib/macros/mm[nt]	non-compacted version of the package
/usr/lib/macros/cmp.n.[dt].m	compacted version of the package
/usr/lib/macros/ucmp.n.m	initializers for the compacted version of the package

SEE ALSO

mm(1), mmt(1), nroff(1), troff(1).

NAME

mmlint − sroff/MM nroff/MM document compatibility checker

SYNOPSIS

mmlint −s file
mmlint −n file

DESCRIPTION

Mmlint reads *file* (an input document) and reports the document changes needed to convert the document to be runnable by the text formatter specified by the option.

−s *mmlint* will flag nroff/MM constructs that are illegal in sroff/MM.

−n *mmlint* will flag sroff/MM constructs that are illegal in nroff/MM.

Constructs are commands, embedded commands, or register references.

There are three types of messages:

Equivalent messages,
which give the equivalent construct in the target formatter.

Non-equivalent messages,
which indicate that there is no equivalent construct in the target formatter.

Warning messages,
which describe the different meanings of a command or argument in each formatter.

Messages are output on standard output.

CAVEATS

With the -s option, *mmlint* assumes the input file is in *nroff/MM* format. However, if the file is in *sroff/MM* format, some erroneous messages may appear. For example,

\(ad\(asr)): no special chars in sroff

although this is a legal register construct in *sroff*.

The same characteristic is true for the **−n** option, with the following messages:

\(sl): use \n(sl) in nroff

although in *nroff*, this is the character sequence "**/)**".

\t: use \nt in nroff

although in *nroff*, **\t** is the tab escape sequence.

\(:Mu): register names can only be two characters long in nroff

although **:M** is a legal register name in *nroff*.

.so and **.nx** requests are ignored by **mmlint.**

NAME

mmt, mvt − typeset documents, viewgraphs, and slides

SYNOPSIS

mmt [options] [files]

mvt [options] [files]

DESCRIPTION

These two commands are very similar to *mm*(1), except that they both typeset their input via *troff*(1), as opposed to formatting it via *nroff*(1); *mmt* uses the MM macro package, while *mvt* uses the Macro Package for View Graphs and Slides. These two commands have options to specify preprocessing by *tbl*(1) and/or *pic*(1) and/or *eqn*(1). The proper pipelines and the required arguments and flags for *troff*(1) and for the macro packages are generated, depending on the options selected.

Options are given below. Any other arguments or flags (e.g., **−rC3**) are passed to *troff*(1) or to the macro package, as appropriate. Such options can occur in any order, but they must appear before the *files* arguments. If no arguments are given, these commands print a list of their options.

−e Causes these commands to invoke *eqn*(1); also causes *eqn* to read the **/usr/pub/eqnchar** file (see *eqnchar*(5)).

−t Causes these commands to invoke *tbl*(1).

−p Invokes *pic*(1).

−Taps Creates output for an Autologic APS-5 phototypesetter, and sends it to the default destination at this installation.

−T*dest* Creates output for *troff* device *dest* (see *troff*(1)). The output is sent through the appropriate postprocessor (see *daps*(1)).

−Tcat Uses *otroff*(1) to generate output for an on-line Wang CAT phototypesetter.

−D4014 Directs the output to a TEKTRONIX 4014 terminal via the *tc*(1) filter.

−Dtek Same as **−D4014**.

−Dli10 Directs the output to the local Imagen Imprint-10 laser printer.

−a Invokes the **−a** option of *troff*(1).

−y Causes *mmt* to use the non-compacted version of the macros. This is the default except when using **−Tcat**.

−z Invokes no output filter to process or redirect the output of *troff*(1).

These commands read the standard input when **−** is specified instead of any file names.

Mvt is just a link to *mmt*.

HINT

Use the **−o***list* option of *troff*(1) to specify ranges of pages to be output. Note, however, that these commands, if invoked with one or more of the **−e**, **−t**, and **−** options, *together* with the **−o***list* option of *troff*(1) may cause a harmless "broken pipe" diagnostic if the last page of the document is not specified in *list*.

SEE ALSO

daps(1), env(1), eqn(1), mm(1), nroff(1), pic(1), tbl(1), tc(1), troff(1), profile(4), environ(5), mm(5), mv(5).

DIAGNOSTICS

"m[mv]t: no input file" if none of the arguments is a readable file and the command is not used as a filter.

NAME

mosd – the OSDD adapter macro package for formatting documents

SYNOPSIS

osdd [options] [files]

mm –mosd [options] [files]

nroff –mm –mosd [options] [files]

nroff –cm –mosd [options] [files]

mmt –mosd [options] [files]

troff –mm –mosd [options] [files]

DESCRIPTION

The OSDD adapter macro package is a tool used in conjunction with the MM macro package to prepare Operations Systems Deliverable Documentation. Many of the OSDD Standards are different from the default format provided by MM. The OSDD adapter package sets the appropriate MM options for automatic production of the OSDD Standards. The OSDD adapter package also generates the correct OSDD page headers and footers, heading styles, Table of Contents format, etc.

OSDD document (input) files are prepared with the MM macros. Additional information which must be given at the beginning of the document file is specified by the following string definitions:

```
.ds H1 document-number
.ds H2 section-number
.ds H3 issue-number
.ds H4 date
.ds H5 rating
```

The *document-number* should be of the standard 10-character format. The words "Section" and "Issue" should not be included in the string definitions; they will be supplied automatically when the document is printed. For example:

```
.ds H1 OPA–1P135–01
.ds H2 4
.ds H3 2
```

automatically produces

```
OPA-1P135-01
Section 4
Issue 2
```

as the document page header. Quotation marks are not used in string definitions.

If certain information is not to be included in a page header, then the string is defined as null; e.g.,

```
.ds H2
```

means that there is no *section-number*.

The OSDD Standards require that the *Table of Contents* be numbered beginning with *Page 1*. By default, the first page of text will be numbered *Page 2*. If the *Table of Contents* has more than one page, for example *n*, then either **–rP***n*+*1* must be included as a command line option or **.nr P** *n* must be included in the document file. For example, if the *Table of Contents* is four pages then use **–rP5** on the command line or **.nr P 4** in the document file.

The OSDD Standards require that certain information such as the document *rating* appear on the *Document Index* or on the *Table of Contents* page if there is no index. By default, it is assumed that an index has been prepared

separately. If there is no index, the following must be included in the document file:

```
.nr Di 0
```

This will ensure that the necessary information is included on the *Table of Contents* page.

The OSDD Standards require that all numbered figures be placed at the end of the document. The **.Fg** macro is used to produce full page figures. This macro produces a blank page with the appropriate header, footer, and figure caption. Insertion of the actual figure on the page is a manual operation. The macro usage is

```
.Fg page-count "figure caption"
```

where *page-count* is the number of pages required for a multi-page figure (default 1 page).

The **.Fg** macro cannot be used within the document unless the final **.Fg** in a series of figures is followed by a **.SK** macro to force out the last figure page.

The *Table of Contents* for OSDD documents (see Figure 4 in Section 4.1 of the OSDD Standards) is produced with:

```
.Tc
System Type
System Name
Document Type
.Td
```

The **.Tc**/**.Td** macros are used instead of the .TC macro from MM.

The **.PM** macro may be used to generate proprietary markings – see the MM document for legal styles.

The **.P** macro is used for paragraphs. The **Np** register is set automatically to indicate the paragraph numbering style. It is very important that the **.P** macro be used correctly. All paragraphs (including those immediately following a **.H** macro) must use a **.P** macro. Unless there is a **.P** macro, there will not be a number generated for the paragraph. Similarly, the **.P** macro should not be used for text which is not a paragraph. The **.SP** macro may be appropriate for these cases, e.g., for "paragraphs" within a list item.

The page header format is produced automatically in accordance with the OSDD Standards. The OSDD Adapter macro package uses the **.TP** macro for this purpose. Therefore the **.TP** macro normally available in MM is not available for users.

FILES

/usr/lib/tmac/tmac.osd

SEE ALSO

mm(1), mmt(1), nroff(1), troff(1), mm(5).

NAME

mptx – the macro package for formatting a permuted index

SYNOPSIS

nroff –mptx [options] [files]

troff –mptx [options] [files]

DESCRIPTION

This package provides a definition for the **.xx** macro used for formatting a permuted index as produced by *ptx*(1). This package does not provide any other formatting capabilities such as headers and footers. If these or other capabilities are required, the *mptx* macro package may be used in conjuction with the *MM* macro package. In this case, the **–mptx** option must be invoked *after* the **–mm** call. For example:

```
nroff –cm –mptx file
```

or

```
mm –mptx file
```

FILES

/usr/lib/tmac/tmac.ptx	pointer to the non-compacted version of the package
/usr/lib/macros/ptx	non-compacted version of the package

SEE ALSO

mm(1), nroff(1), ptx(1), troff(1), mm(5).

NAME

mv – a troff macro package for typesetting viewgraphs and slides

SYNOPSIS

mvt [**–a**] [options] [files]

troff [**–a**] [**–rX1**] **–mv** [options] [files]

DESCRIPTION

This package makes it easy to typeset viewgraphs and projection slides in a variety of sizes. A few macros (briefly described below) accomplish most of the formatting tasks needed in making transparencies. All of the facilities of *troff*(1), *eqn*(1), and *tbl*(1) are available for more difficult tasks.

The output can be previewed on most terminals, and, in particular, on the TEKTRONIX 4014. For this device, specify the **–rX1** option (this option is automatically specified by the *mvt* command–q.v.–when that command is invoked with the **–T4014** option). To preview output on other terminals, specify the **–a** option.

The available macros are:

.VS [*n*] [*i*] [*d*] Foil-start macro; foil size is to be 7″×7″; *n* is the foil number, *i* is the foil identification, *d* is the date; the foil-start macro resets all parameters (indent, point size, etc.) to initial default values, except for the values of *i* and *d* arguments inherited from a previous foil-start macro; it also invokes the .A macro (see below).

The naming convention for this and the following eight macros is that the first character of the name (**V** or **S**) distinguishes between viewgraphs and slides, respectively, 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 "skinnier" than the corresponding viewgraphs: 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.

.Vw [*n*] [*i*] [*d*] Same as **.VS**, except that foil size is 7″ wide × 5″ high.
.Vh [*n*] [*i*] [*d*] Same as **.VS**, except that foil size is 5″×7″.
.VW [*n*] [*i*] [*d*] Same as **.VS**, except that foil size is 7″×5.4″.
.VH [*n*] [*i*] [*d*] Same as **.VS**, except that foil size is 7″×9″.
.Sw [*n*] [*i*] [*d*] Same as **.VS**, except that foil size is 7″×5″.
.Sh [*n*] [*i*] [*d*] Same as **.VS**, except that foil size is 5″×7″.
.SW [*n*] [*i*] [*d*] Same as **.VS**, except that foil size is 7″×5.4″.
.SH [*n*] [*i*] [*d*] Same as **.VS**, except that foil size is 7″×9″.

.A [*x*] Place text that follows at the first indentation level (left margin); the presence of *x* suppresses the ½ line spacing from the preceding text.

.B [*m* [*s*]] Place text that follows at the second indentation level; text is preceded by a mark; *m* is the mark (default is a large bullet); *s* is the increment or decrement to the point size of the mark with respect to the *prevailing* point size (default is 0); if *s* is 100, it causes the point size of the mark to be the same as that of the *default* mark.

.C [*m* [*s*]] Same as **.B**, but for the third indentation level; default mark is a dash.

.D [*m* [*s*]] Same as **.B**, but for the fourth indentation level; default mark is a small bullet.

.T *string* *String* is printed as an over-size, centered title.

.I [*in*] [*a* [*x*]] Change the current text indent (does not affect titles); *in* is the indent (in inches unless dimensioned, default is 0); if *in* is signed, it is an increment or decrement; the presence of *a* invokes the **.A** macro (see below) and passes *x* (if any) to it.

.S [*p*] [*l*] Set the point size and line length; *p* is the point size (default is "previous"); if *p* is 100, the point size reverts to the *initial* default for the current foil-start macro; if *p* is signed, it is an increment or decrement (default is 18 for **.VS**, **.VH**, and **.SH**, and 14 for the other foil-start macros); *l* is the line length (in inches unless dimensioned; default is 4.2" for **.Vh**, 3.8" for **.Sh**, 5" for **.SH**, and 6" for the other foil-start macros).

.DF *n f* [*n f ...*] Define font positions; may not appear within a foil's input text (i.e., it may only appear after all the input text for a foil, but before the next foil-start macro); *n* is the position of font *f*; up to four "*n f*" pairs may be specified; the first font named becomes the *prevailing* font; the initial setting is (**H** is a synonym for **G**):

.DF 1 H 2 I 3 B 4 S

.DV [*a*] [*b*] [*c*] [*d*] Alter the vertical spacing between indentation levels; *a* is the spacing for **.A**, *b* is for **.B**, *c* is for **.C**, and *d* is for **.D**; all non-null arguments must be dimensioned; null arguments leave the corresponding spacing unaffected; initial setting is:

.DV .5v .5v .5v 0v

.U *str1* [*str2*] Underline *str1* and concatenate *str2* (if any) to it.

The last four macros in the above list do not cause a break; the **.I** macro causes a break only if it is invoked with more than one argument; all the other macros cause a break.

The macro package also recognizes the following upper-case synonyms for the corresponding lower-case *troff* requests:

.AD .BR .CE .FI .HY .NA .NF .NH .NX .SO .SP .TA .TI

The **Tm** string produces the trademark symbol.

The input tilde (~) character is translated into a blank on output.

See the user's manual cited below for further details.

FILES

/usr/lib/tmac/tmac.v
/usr/lib/macros/vmca

SEE ALSO

eqn(1), mmt(1), tbl(1), troff(1).

BUGS

The **.VW** and **.SW** foils are meant to be 9" wide by 7" high, but because the typesetter paper is generally only 8" wide, they are printed 7" wide by 5.4" high and have to be enlarged by a factor of 9/7 before use as viewgraphs; this makes them less than totally useful.

NAME

non-btl – reinstall MM macros without Bell Laboratories specific features

SYNOPSIS

sh non-btl.sh

DESCRIPTION

The *non-btl.sh* command will modify and re-install the source for the Memorandum Macros (used with *nroff* and *troff*) when Bell Labs specific macros are not desired.

Specifically, use of the *non-btl.sh* command will remove the .TM, .PM, .CS macros, and the }2 string (which normally contains the name "Bell Laboratories") from the macro package. After running *non-btl.sh*, use of these features will have no effect.

This command does not remove the source for these features from the macro file, but does erase their definition. Those users who wish to tailor the macro package to their own environment may choose not to run *non-btl.sh*, but to modify the definition of the affected macros and string to their own specifications. Remember to re-install the macros after they are modified.

IMPORTANT

The *non-btl.sh* command is found in the **/usr/src/cmd/text/macros.d** directory, and may only be run by the super-user.

NAME

nroff, otroff – format or typeset text

SYNOPSIS

nroff [options] [files]

otroff [options] [files]

DESCRIPTION

Norff formats text contained in *files* (standard input by default) for printing on typewriter-like devices and line printers; similarly, *otroff* formats text for a Wang Laboratories, Inc., C/A/T phototypesetter. Their capabilities are described in the "Nroff and Troff User Manual".

An argument consisting of a minus (**–**) is taken to be a file name corresponding to the standard input. The *options*, which may appear in any order, but must appear before the *files*, are:

–o*list* Print only pages whose page numbers appear in the *list* of numbers and ranges, separated by commas. A range *N*–*M* means pages *N* through *M*; an initial –*N* means from the beginning to page *N*; and a final *N*– means from *N* to the end. (See *BUGS* below.)

–n*N* Number first generated page *N*.

–s*N* Stop every *N* pages. *Nroff* will halt *after* every *N* pages (default *N*=1) to allow paper loading or changing, and will resume upon receipt of a line-feed or new-line (new-lines do not work in pipelines, e.g., with *mm*(1)). This option does not work if the output of *nroff* is piped through *col*(1). *Otroff* will stop the phototypesetter every *N* pages, produce a trailer to allow changing cassettes, and resume when the typesetter's start button is pressed. When *nroff* (*otroff*) halts between pages, an ASCII **BEL** (in *otroff*, the message **page stop)** is sent to the terminal.

–r*aN* Set register *a* (which must have a one-character name) to *N*.

–i Read standard input after *files* are exhausted.

–q Invoke the simultaneous input-output mode of the **.rd** request.

–z Print only messages generated by **.tm** (terminal message) requests.

–m*name* Prepend to the input *files* the non-compacted (ASCII text) macro file **/usr/lib/tmac/tmac.***name*.

–c*name* Prepend to the input *files* the compacted macro files **/usr/lib/macros/cmp.[nt].[dt].***name* and **/usr/lib/macros/ucmp.[nt].***name*.

–k*name* Compact the macros used in this invocation of *nroff*/*otroff*, placing the output in files **[dt].***name* in the current directory (see the *DOCUMENTERS WORKBENCH Software Text Formatters Reference* for details of compacting macro files).

Nroff only:

–T*name* Prepare output for specified terminal. Known *names* are **37** for the (default) TELETYPE® Model 37 terminal, **tn300** for the GE TermiNet 300 (or any terminal without half-line capability), **300s** for the DASI 300s, **300** for the DASI 300, **450** for the DASI 450, **lp** for a (generic) ASCII line printer, **382** for the DTC-382, **4000A** for the Trendata 4000A, **832** for the Anderson Jacobson 832, **X** for a (generic) EBCDIC printer, and **2631** for the Hewlett Packard 2631 line printer.

–e Produce equally-spaced words in adjusted lines, using the full resolution of the particular terminal.

–h Use output tabs during horizontal spacing to speed output and reduce output character count. Tab settings are assumed to be

every 8 nominal character widths.

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

Otroff only:

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

−f Refrain from feeding out paper and stopping phototypesetter at the end of the run.

−w Wait until phototypesetter is available, if it is currently busy.

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

−a Send a printable ASCII approximation of the results to the standard output.

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

−Tcat Use font-width tables for Wang CAT phototypesetter. This device is both the default and the only choice.

FILES

/usr/lib/suftab	suffix hyphenation tables
/tmp/ta$#	temporary file
/usr/lib/tmac/tmac.*	standard macro files and pointers
/usr/lib/macros/*	standard macro files
/usr/lib/term/*	terminal driving tables for *nroff*
/usr/lib/font/*	font width tables for *otroff*

SEE ALSO

eqn(1), ocw(1), tbl(1), mm(5).

480.sp0u

nroff only-

col(1), greek(1), mm(1).

otroff only-

mmt(1), mv(5).

BUGS

Nroff/otroff believes in Eastern Standard Time; as a result, depending on the time of the year and on your local time zone, the date that *nroff/otroff* generates may be off by one day from your idea of what the date is.

When *nroff/otroff* is used with the **−o***list* option inside a pipeline (e.g., with one or more of *ocw*(1), *eqn*(1), and *tbl*(1)), it may cause a harmless "broken pipe" diagnostic if the last page of the document is not specified in *list*.

NAME

ocw, checkcw — prepare constant-width text for otroff

SYNOPSIS

ocw [**-lxx**] [**-rxx**] [**-fn**] [**-t**] [**+t**] [**-d**] [files]

checkcw [**-lxx**] [**-rxx**] files

DESCRIPTION

Ocw is a preprocessor for *otroff* (see *nroff*(1)) input files that contain text to be typeset in the constant-width (CW) font on the Wang CAT phototypesetter. This preprocessor is not necessary for users of the new device-independent *troff*(1), nor is it compatible with it. Refer to the Addendum to the NROFF/TROFF User Manual for details on how to eliminate the use of this command.

C-2

Text typeset with the CW font resembles the output of terminals and of line printers. This font is used to typeset examples of programs and of computer output in user manuals, programming texts, etc. (An earlier version of this font was used in typesetting *The C Programming Language* by B. W. Kernighan and D. M. Ritchie.) It has been designed to be quite distinctive (but not overly obtrusive) when used together with the Times Roman font.

Because the CW font on the Wang CAT contains a "non-standard" set of characters and because text typeset with it requires different character and interword spacing than is used for "standard" fonts, documents that use the CW font must be preprocessed by *ocw*.

The CW font contains the 94 printing ASCII characters:

```
abcdefghijklmnopqrstuvwxyz
ABCDEFGHIJKLMNOPQRSTUVWXYZ
0123456789
!$%&()`'*+@.,/:;=?[]¦-_^~"<>{}#\
```

plus eight non-ASCII characters represented by four-character *otroff* names (in some cases attaching these names to "non-standard" graphics):

Character	*Symbol*	*Troff Name*
"Cents" sign	¢	\(ct
EBCDIC "not" sign	¬	\(no
Left arrow	←	\(<-
Right arrow	→	\(->
Down arrow	↓	\(da
Vertical single quote	'	\(fm
Control-shift indicator	^	\(dg
Visible space indicator	␣	\(sq
Hyphen	-	\(hy

The hyphen is a synonym for the unadorned minus sign (-). Certain versions of *ocw* recognize two additional names: `\(ua` for an up arrow and `\(lh` for a diagonal left-up (home) arrow.

Ocw recognizes five request lines, as well as user-defined delimiters. The request lines look like *otroff* macro requests, and are copied in their entirety by *ocw* onto its output; thus, they can be defined *by the user* as *otroff* macros; in fact, the .CW and .CN macros *should* be so defined (see *HINTS* below). The five requests are:

.CW Start of text to be set in the CW font; .CW causes a break; it can take precisely the same options, in precisely the same format, as are available on the *ocw* command line.

.CN End of text to be set in the CW font; .CN causes a break; it can take the same options as are available on the *ocw* command line.

.CD Change delimiters and/or settings of other options; takes the same options as are available on the *ocw* command line.

.CP *arg1 arg2 arg3 ... argn*
All the arguments (which are delimited like *otroff* macro arguments) are concatenated, with the odd-numbered arguments set in the CW font and the even-numbered ones in the prevailing font.

.PC *arg1 arg2 arg3 ... argn*
Same as .CP, except that the even-numbered arguments are set in the CW font and the odd-numbered ones in the prevailing font.

The .CW and .CN requests are meant to bracket text (e.g., a program fragment) that is to be typeset in the CW font "as is." Normally, *ocw* operates in the *transparent* mode. In that mode, except for the .CD request and the nine special four-character names listed in the table above, every character between .CW and .CN request lines stands for itself. In particular, *ocw* arranges for periods (.) and apostrophes (') at the beginning of lines, and backslashes (\) everywhere to be "hidden" from *otroff*. The transparent mode can be turned off (see below), in which case normal *otroff* rules apply; in particular, lines that begin with . and ' are passed through untouched (except if they contain delimiters—see below). In either case, *ocw* hides the effect of the font changes generated by the .CW and .CN requests; *ocw* also defeats all ligatures (fi, ff, etc.) in the CW font.

The only purpose of the .CD request is to allow the changing of various options other than just at the beginning of a document.

The user can also define *delimiters*. The left and right delimiters perform the same function as the .CW/.CN requests; they are meant, however, to enclose CW "words" or "phrases" in running text (see example under *BUGS* below). *Ocw* treats text between delimiters in the same manner as text enclosed by .CW/.CN pairs, except that, for aesthetic reasons, spaces and backspaces inside .CW/.CN pairs have the same width as other CW characters. While spaces and backspaces between delimiters are half as wide, so they have the same width as spaces in the prevailing text (but are *not* adjustable). Font changes due to delimiters are *not* hidden.

Delimiters have no special meaning inside .CW/.CN pairs.

The options are:

-l*xx* The one- or two-character string *xx* becomes the left delimiter; if *xx* is omitted, the left delimiter becomes undefined, which it is initially.

-r*xx* Same for the right delimiter. The left and right delimiters may (but need not) be different.

-f*n* The CW font is mounted in font position *n*; acceptable values for *n* are 1, 2, and 3 (default is 3, replacing the bold font). This option is only useful at the beginning of a document.

-t Turn transparent mode *off*.

+t Turn transparent mode *on* (this is the initial default).

-d Print current option settings on file descriptor 2 in the form of *otroff* comment lines. This option is meant for debugging.

Ocw reads the standard input when no *files* are specified (or when - is specified as the last argument), so it can be used as a filter. Typical usage is: ocw *files* | otroff ... *Checkcw* checks that left and right delimiters, as well as the .CW/.CN pairs, are properly balanced. It prints out all offending lines.

HINTS

Typical definitions of the .CW and .CN macros meant to be used with the *mm*(5) macro package:

```
.de CW
.DS I
.ps 9
.vs 10.5p
.ta 16m/3u 32m/3u 48m/3u 64m/3u 80m/3u 96m/3u ...
..
.de CN
.ta 0.5i 1i 1.5i 2i 2.5i 3i 3.5i 4i 4.5i 5i 5.5i 6i
.vs
.ps
.DE
..
```

At the very least, the .CW macro should invoke the *otroff* no-fill (.nf) mode.

When set in running text, the CW font is meant to be set in the same point size as the rest of the text. In displayed matter, on the other hand, it can often be profitably set one point *smaller* than the prevailing point size (the displayed definitions of .CW and .CN above are one point smaller than the running text on this page). The CW font is sized so that, when it is set in 9-point, there are 12 characters per inch.

Documents that contain CW text may also contain tables and/or equations. If this is the case, the order of preprocessing should be: *ocw*, *tbl*, and *eqn*. Usually, the tables contained in such documents will not contain any CW text, although it is entirely possible to have *elements* of the table set in the CW font; of course, care must be taken that *tbl*(1) format information not be modified by *ocw*. Attempts to set equations in the CW font are not likely to be either pleasing or successful.

In the CW font, overstriking is most easily accomplished with backspaces: letting ← represent a backspace, `d←←\(dg` yields d̂. (Because backspaces are half as wide between delimiters as inside .CW/.CN pairs—see above—two backspaces are required for each overstrike between delimiters.)

FILES

/usr/lib/font/ftCW CW font-width table

SEE ALSO

eqn(1), nroff(1), tbl(1), mm(5), mv(5).

WARNINGS

If text preprocessed by *ocw* is to make any sense, it must be set on a typesetter equipped with the CW font or on a STARE facility; on the latter, the CW font appears as bold, but with the proper CW spacing.

BUGS

Only a masochist would use periods (.), backslashes (\), or double quotes (") as delimiters, or as arguments to .CP and .PC.

Certain CW characters do not concatenate gracefully with certain Times Roman characters, e.g., a CW ampersand (&) followed by a Times Roman comma(,). In such cases, judicious use of *otroff* half- and quarter-spaces (\| and \^) is most salutary, e.g., one should use _&_\^, (rather than just plain _&_,) to obtain &, (assuming that _ is used for both delimiters).

Using *ocw* with *nroff* is silly.

NAME

pic — troff preprocessor for drawing simple pictures

SYNOPSIS

pic [**−T***t*] [files]

DESCRIPTION

Pic is a *troff*(1) preprocessor for drawing simple figures on a typesetter. The basic objects are *box*, *line*, *arrow*, *circle*, *ellipse*, *arc* and text.

The optional argument **−T***t* specifies device *t*; currently supported devices are **aps** (Autologic APS-5), **X97** (Xerox 9700), and **i10** (Imagen Imprint-10). Default is **−Taps**.

SEE ALSO

troff(1).

PIC — A Graphics Language for Typesetting.

NAME

sroff – format text

SYNOPSIS

sroff [options] [files]

DESCRIPTION

Sroff formats text contained in *files* (standard input by default) for printing on typewriter-like devices and line printers, including the XEROX 9700 printer.

An argument consisting of a minus (–) is taken to be a file name corresponding to the standard input. The **options**, which may appear in any order, but must appear before the *files*, are:

–o*list* Print only pages whose page numbers appear in the *list* of numbers and ranges, separated by commas. A range *N*–*M* means pages *N* through *M*; an initial –*N* means from the beginning to page *N*; and a final *N*– means from *N* to the end.

–s*N* Stop every *N* pages. *Sroff* will halt *after* every *N* pages (default *N*=1) to allow paper loading or changing, and will resume upon receipt of a line-feed or new-line.

–m*name* Prepend to the input *files* the macro file **/usr/lib/smac/m***name*. (None available so far. Development of an MM-like macro package for *sroff* is in progress.)

–x*file* Write any index information onto *file*.

SEE ALSO

col(1), pg(1).

BUGS

%# is the name of a register that contains the number of lines used on a page in single-column mode, or the number of lines in a diversion. %# should work in multi-column mode, but what should it count?

NAME

tbl – format tables for nroff or troff

SYNOPSIS

tbl [**–TX**] [files]

DESCRIPTION

Tbl is a preprocessor that formats tables for *nroff* or *troff*(1). The input files are copied to the standard output, except for lines between .TS and .TE command lines, which are assumed to describe tables and are re-formatted by *tbl*. (The .TS and .TE command lines are not altered by *tbl*).

.TS is followed by global options. The available global options are:

center	center the table (default is left-adjust);
expand	make the table as wide as the current line length;
box	enclose the table in a box;
doublebox	enclose the table in a double box;
allbox	enclose each item of the table in a box;
tab (*x*)	use the character *x* instead of a tab to separate items in a line of input data.

The global options, if any, are terminated with a semi-colon (;).

Next come lines describing the format of each line of the table. Each such format line describes one line of the actual table, except that the last format line (which must end with a period) describes *all* remaining lines of the table. Each column of each line of the table is described by a single key-letter, optionally followed by specifiers that determine the font and point size of the corresponding item, that indicate where vertical bars are to appear between columns, that determine column width, inter-column spacing, etc. The available key-letters are:

c	center item within the column;
r	right-adjust item within the column;
l	left-adjust item within the column;
n	numerically adjust item in the column: units positions of numbers are aligned vertically;
s	span previous item on the left into this column;
a	center longest line in this column and then left-adjust all other lines in this column with respect to that centered line;
^	span down previous entry in this column;
_	replace this entry with a horizontal line;
=	replace this entry with a double horizontal line.

The characters **B** and **I** stand for the bold and italic fonts, respectively; the character | indicates a vertical line between columns.

The format lines are followed by lines containing the actual data for the table, followed finally by .TE. Within such data lines, data items are normally separated by tab characters.

If a data line consists of only _ or =, a single or double line, respectively, is drawn across the table at that point; if a *single item* in a data line consists of only _ or =, then that item is replaced by a single or double line.

Full details of all these and other features of *tbl* are given in the reference manual cited below.

The **–TX** option forces *tbl* to use only full vertical line motions, making the output more suitable for devices that cannot generate partial vertical line motions (e.g., line printers).

If no file names are given as arguments (or if − is specified as the last argument), *tbl* reads the standard input, so it may be used as a filter. When it is used with *eqn*(1) or *neqn*, *tbl* should come first to minimize the volume of data passed through pipes.

EXAMPLE

If we let → represent a tab (which should be typed as a genuine tab), then the input:

```
.TS
center box ;
cB s s
cl | cl s
^ | c c
l | n n .
Household Population

_
Town→Households
→Number→Size
=
Bedminster→789→3.26
Bernards Twp.→3087→3.74
Bernardsville→2018→3.30
Bound Brook→3425→3.04
Bridgewater→7897→3.81
Far Hills→240→3.19
.TE
```

yields:

Household Population		
Town	*Households* Number	Size
Bedminster	789	3.26
Bernards Twp.	3087	3.74
Bernardsville	2018	3.30
Bound Brook	3425	3.04
Bridgewater	7897	3.81
Far Hills	240	3.19

SEE ALSO

ocw(1), eqn(1), mm(1), mmt(1), nroff(1), troff(1), mm(5), mv(5).

BUGS

See *BUGS* under *nroff*(1).

NAME

tc, otc – troff output interpreter

SYNOPSIS

tc [**–t**] [**–olist**] [**–an**] [**–e**] [file]

otc [**–t**] [**–sn**] [**–pl**] [file]

DESCRIPTION

Tc interprets its input (standard input default) as output from *troff*(1). The standard output of *tc* is intended for a TEKTRONIX 4015 (a 4014 terminal with ASCII and APL character sets). The various typesetter sizes are mapped into the 4014's four sizes; the entire TROFF character set is drawn using the 4014's character generator, using overstruck combinations where necessary, producing an altogether displeasing effect. *Otc* performs a similar function for the old TROFF, *otroff* (see *nroff*(1)). Typical usage:

troff file | tc

otroff -t file | otc

At the end of each page *tc* waits for a new-line (empty line) from the keyboard before continuing on to the next page. In this wait state, the following commands are recognized:

!*cmd* Send *cmd* to the shell.

e Invert state of the screen erase (*tc*); do not erase screen before next page (*otc*).

–n Skip backward *n* pages. (*tc* only).

n Print page *n*. (*tc* only).

s*n* Skip forward *n* pages. (*otc* only).

a*n* Set the aspect ratio to *n*. (*tc* only).

? Print list of available options. (*tc* only).

The command line options are:

–t Do not wait between pages (for directing output into a file).

–o*list* Prints only the pages enumerated in *list*. The list consists of pages and page ranges (e.g., 5-17) separated by commas. The range *n*– goes from *n* to the end; the range *–n* goes from the beginning to and including page *n*. (*tc* only).

–a*n* Set the aspect ratio to *n*; default is 1.5. (*tc* only).

–e Do not erase before each page. (*tc* only).

–s*n* Skip the first *n* pages. (*otc* only).

–p*l* Set page length to *l*; *l* may include the scale factors **p** (points), **i** (inches), **c** (centimeters), and **P** (picas); Default is picas. (*otc* only).

SEE ALSO

4014(1), nroff(1), tplot(1G), troff(1).

BUGS

Font distinctions are lost.

It needs a **–w** option to wait for input to arrive.

NAME

troff – text formatting and typesetting

SYNOPSIS

troff [option] ... [file] ...

DESCRIPTION

Troff formats text in the named *files* for printing on a phototypesetter. It is the new "device-independent" version of the old *otroff* (see *nroff*(1)). Its capabilities are described in the "Nroff and Troff User Manual".

If no *file* argument is present, the standard input is read. An argument consisting of a single minus (–) is taken to be a file name corresponding to the standard input. The options, which may appear in any order so long as they appear before the files, are:

–o*list* Print only pages whose page numbers appear in the comma-separated *list* of numbers and ranges. A range *N*–*M* means pages *N* through *M*; an initial –*N* means from the beginning to page *N*; and a final *N*– means from *N* to the end. (See BUGS below.)

–n*N* Number first generated page *N*.

–s*N* Generate output to encourage typesetter to stop every *N* pages.

–m*name* Prepend the macro file **/usr/lib/tmac/tmac.***name* to the input *files*.

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

–i Read standard input after the input files are exhausted.

–q Invoke the simultaneous input-output mode of the **.rd** request.

–z Print only messages generated by **.tm** requests.

–a Send a printable ASCII approximation of the results to the standard output.

–T*dest* Prepare output for typesetter *dest*. Currently the only supported typesetter is the Autologic APS-5, (**–Taps**). Users of the Wang CAT should use *otroff* (see *nroff*(1)). Supported laser printers are the Imagen Imprint -10 (**–T***i10*) and the Xerox 9700 (see *dx9700*(1)).

FILES

/tmp/trtmp*	temporary file
/usr/lib/tmac/tmac.*	standard macro files
/usr/lib/macros/*	standard macro files
/usr/lib/font/dev*/*	font width tables

SEE ALSO

daps(1), dx9700(1), eqn(1), mmt(1), nroff(1), pic(1), tbl(1), tc(1).

"Nroff and Troff User Manual"

BUGS

The **.tl** request may not be used before the first break-producing request in the input to *troff*.

Troff believes in Eastern Standard Time; as a result, depending on the time of the year and on your local time zone, the date that *troff* generates may be off by one day from your idea of what the date is.

When *troff* is used with the –o*list* option inside a pipeline (e.g., with one or more of *pic*(1), *eqn*(1), and *tbl*(1)), it may cause a harmless "broken pipe" diagnostic if the last page of the document is not specified in *list*.

NAME

troff – description of output language

DESCRIPTION

The device-independent *troff* outputs a pure ASCII description of a typeset document. The description specifies the typesetting device, the fonts, and the point sizes of characters to be used as well as the position of each character on the page. A list of all the legal commands follows. Most numbers are denoted as *n* and are ASCII strings. Strings inside of [ ] are optional. *Troff* may produce them, but they are not required for the specification of the language. The character **\n** has the standard meaning of "newline" character. Between commands white space has no meaning. White space characters are spaces and newlines. All commands which have an arbitary length numerical parameter or word must be followed by white space. For example, the command to specify point size, s###, must be followed by a space or newline.

s*n*	The point size of the characters to be generated.
f*n*	The font mounted in the specified position is to be used. The number ranges from 0 to the highest font presently mounted. 0 is a special position, invoked by *troff*, but not directly accessible to the troff user. Normally fonts are mounted starting at position 1.
c*x*	Generate the character *x* at the current location on the page; *x* is a single ASCII character.
C*xyz*	Generate the special character *xyz*. The name of the character is delimited by white space. The name will be one of the special characters legal for the typesetting device as specified by the device specification found in the file *DESC*. This file resides in a directory specific for the typesetting device. (See *font*(5) and **/usr/lib/font/dev***.)
H*n*	Change the horizonal position on the page to the number specified. The number is in basic units of motions as specified by *DESC*. This is an absolute "goto".
h*n*	Add the number specified to the current horizontal position. This is a relative "goto".
V*n*	Change the vertical position on the page to the number specified (down is positive).
v*n*	Add the number specified to the current vertical position.
nnx	This is a two-digit number followed by an ASCII character. The meaning is a combination of h*n* followed by c*x*. The two digits *nn* are added to the current horizontal position and then the ASCII character, *x*, is produced. This is the most common form of character specification.
n*b a*	This command indicates that the end of a line has been reached. No action is required, though by convention the horizontal position is set to 0. *Troff* will specify a resetting of the *x*,*y* coordinates on the page before requesting that more characters be printed. The first number, *b*, is the amount of space before the line and the second number, *a*, the amount of space after the

line. The second number is delimited by white space.

w — A w appears between words of the input document. No action is required. It is included so that one device can be emulated more easily on another device.

p*n* — Begin a new page. The new page number is included in this command. The vertical position on the page should be set to 0.

{ — Push the current environment, which means saving the current point size, font, and location on the page.

} — Pop a saved environment.

t*xxxxx* — Print the string of characters, *xxxxx*, using the natural width of each character to determine the next x coordinate. *Troff* does not currently produce this form of command. It is not recommended. The characters will probably be too close together.

 # .... \n — A line beginning with a pound sign is a comment.

Dl *x y***\n** — Draw a line from the current location to *x,y*. At the end of the drawing operation the current location will be *x,y*.

Dc *d***\n** — Draw a circle of diameter *d* with the leftmost edge being at the current location (x, y). The current location after drawing the circle will be x+*d*,y, the rightmost edge of the circle.

De *dx dy***\n** — Draw an ellipse with the specified axes; *dx* is the axis in the x direction and *dy* is the axis in the y direction. The leftmost edge of the ellipse will be at the current location. After drawing the ellipse the current location will be x+*dx*,y.

Da *x y r***\n** — Draw a counterclockwise arc from the current location to *x,y* using a circle of radius *r* . The current location after drawing the arc will be *x,y*.

D~ *x y x y*...**\n** — Draw a spline curve (wiggly line) between each of the *x,y* coordinate pairs starting at the current location. The final location will be the final *x,y* pair of the list. Currently there may be no more than 36 *x,y* pairs to this command.

x i[nit]\n — Initialize the typesetting device. The actions required are dependent on the device. An **init** command will always occur before any output generation is attempted.

x T *device***\n** — The name of the typesetter is *device*. This is the same as the argument to the **−T** option. The information about the typesetter will be found in the directory **/usr/lib/font/dev**{*device*}.

x r[es] *n h v*\n — The resolution of the typesetting device in increments per inch is *n*. Motion in the horizontal direction can take place in units of *h* basic increments. Motion in the vertical direction can take place in units of *v* basic increments. For example, the APS-5 typesetter has a basic resolution of 723 increments per inch and can move in either direction in 723rds of an inch. Its specification is:
x res 723 1 1

x p[ause]\n	Pause. Cause the current page to finish but do not relinquish the typesetter.
x s[top]\n	Stop. Cause the current page to finish and then relinquish the typesetter. Perform any shutdown and bookkeeping procedures required.
x t[railer]\n	Generate a trailer. On some devices no operation is performed.
x f[ont] *n name*\n	Load the font *name* into position *n*.
x H[eight] *n*\n	Set the character height to *n* points. This causes the letters to be elongated or shortened. It does not affect the width of a letter.
x S[lant] *n*\n	Set the slant to *n* degrees. Only some typesetters can do this and not all angles are supported.

C-2

NAME

x9700 - prepare nroff documents for the Xerox 9700 printer

SYNOPSIS

x9700 [**-1**|**-2**] [[**-f**] file] [**-h** indent] [**-v** indent] [**-l** leng] [**-**[**p**|**l**]**k** mask [n]] [[**-o** orient] [**-s** style] [**-T** c] [files]

DESCRIPTION

The *x9700* command reads the named *files* and writes standard output which is suitable to be sent to the Xerox 9700 printer. The special name - means standard input. Each file will begin on a new page. If no files are specified, then *x9700* reads from standard input. Options and their meanings:

−1 print output on one side of the page

−2 print output on both sides of the page

−f *file* Take input from *file*. This option is necessary to process file names which begin with a hyphen.

−h *indent* horizontal indent: offset output *indent* units to the right. A *c* appended to *indent* sets the unit of offset to centimeters; an *i*, sets the unit to inches; neither, sets the unit to character positions. The default indent is zero. Fractional character positions are ignored.

−v *indent* Vertical indent: offset output *indent* lines from top of page. Default is zero.

−l *length* Print *length* lines per page. Defaults for the fonts are given below. A *length* of zero obtains the default.

−lk *mask n*
−pk *mask n*
−k *mask n* Overlay output with preprinted *mask*. The *lk* overlays the mask in landscape orientation; the *pk*, in portrait orientation. The *k* alone uses the current orientation. The default mask is *none*. A number following the mask name specifies the page on which to overlay the mask. If no number follows the mask name, then all pages not specifically named are overlaid with the mask. Available masks are installation-dependant.

−o *orient* Page orientation, either **portrait** or **landscape**, with **port** and **land** respectively, acceptable abbreviations. Each font style has a default, given below. Specifying an empty orientation obtains the default.

−s *style* Select font style. Current possibilities and default values:

style	abbr	default orient	portrait length	portrait width	landscape length	landscape width
elite	elit	port	71	102	51	131
gothic	goth	port	66	85	51	110
goth24		port	33	42	25	55
mini		port	137	131	106	131
pica		port	66	85	51	110
times14		port	46	~90	36	~118
times28		port	23	~45	18	~59
vintage	vint	port	71	102	55	131
vint20		port	35	51	27	66
xerox	xrox	land	99	116	77	131
xerox18		land	44	58	34	75

Note that the lengths and widths are maximum values for a page and make no provision for margins. The ~ indicates approximate widths for proportionally spaced fonts. The default style is *vintage*. Both the style names and their abbreviations are accepted. Not all styles have all fonts, and not all fonts have a full character set (including the full TX train). A summary of available combinations appears below. **Note: these fonts are under development and subject to change without notice.**

−T *c* If and only if *c* is **X**, then *x9700* expects input from *nroff* with the **-TX** option.

Options may be repeated and may appear in any order. The space between an option and its argument may be omitted. The options are cumulative and apply only to succeeding file names. Thus

```
x9700 -o port -h 10 file1 -o land file2
```

prints *file1* in portrait orientation and *file2* in landscape but indents both files by 10 characters.

ESCAPES

The command *X9700* recognizes four control characters (backspace, formfeed, horizontal tab, and carriage return) and the following set of escapes:

escape sequence	meaning	from NROFF
esc X *c*	hyperascii *c* ('*c*'|0200)	
esc esc *c*	hyperascii *c* ('*c*'|0200)	
esc B	bold font	\fB
esc R	Roman font	\fR
esc I	Italic font	\fI
esc L	logo font	
esc D	reverse half-line feed	\u
esc U	half-line feed	\d
esc \n	reverse line feed	
esc si	intensify shading	
esc so	lessen shading	

The half-line motions effect superscripts and subscripts, but the TX train contains only a limited number of these. There are three levels of shading available: dark (character e9), darker (e8), and darkest (c4).

input this column	to get
none	none
\33\17level 1 (dark)	level 1 (dark)
\33\17level 2 (darker)	level 2 (darker)
\33\17level 3 (darkest)	level 3 (darkest)
\33\16back to level 2	back to level 2
\33\16back to level 1	back to level 1
\33\16back to none	back to none

SEE ALSO

nroff(1).

EXCEPTIONS

Lines that exceed the page width are truncated. Page breaks occur not only at the logical end of page (controlled by the -l option), but also at the physical end of page (controlled by the machine). Lines which exceed the latter limit

are usually forced to an extra, overflow page. The number of lines on a page includes the indent of the -v option.

It is difficult to get to all of the *TX* train.

FONT SUMMARY

style	bold-italic port	bold-italic land	graph port	graph land
elite	y	y	n	n
Gothic	y	y	n	n
goth24	n	n	n	n
mini	n	n	n	n
pica	y	y	n	n
times14	n	n	n	n
times28	n	n	n	n
vintage	y	y	y	y
vint20	n	n	y	y
xerox	y	y	n	n
xerox18	n	n	n	n

DIAGNOSTICS

"missing parameter to -option"
"can't open file"
"unsupported style/orientation combination"
"bad mask name"
"bad horizontal indent specification"
"bad page length specification"
"bad vertical indent specification"
Check parameter list.

"page length larger than max"
X9700 has been directed to place more than 140 lines on a page.

"attempt to back off page"
An attempt to field a reverse line feed would cause a return to a previous page.

"file too wide"
X9700 has encountered a line with more than 132 characters on it. This usually happens when input *not* produced with *nroff -TX* is given to *x9700* with the *-TX* option.

"unknown escape sequence"
X9700 has been given an escape sequence which does not correspond to a reverse line feed, a font change, a shade change, or a hyperascii character. Escape sequences are introduced with an ascii *esc* character (octal 33). This usually happens when *-TX* is not supplied to *nroff*.

"too many masks"
X9700 allows a total of only ten separate mask specifications.

"page too dense"
X9700 has encountered a page with too much overprinting. The cause may be too much backspacing or too many font changes. It may be small comfort that even if the *x9700* program could format the page, the Xerox printer would probably fail to print it.

"internal error"
"machine seized"
Get help.

EXAMPLES

The following examples do not include the final pipeline to direct the output to the Xerox 9700 printer, because that is an installation-dependent procedure.

To obtain standard memo format:

```
nroff -rA3 -rE1 -rU1 -rL71 -TX -cm file |
x9700 -h10 -TX -k prin1
```

To obtain manual page:

```
nroff -TX -man file |
x9700 -l66 -v3 -h10 -TX
```

To obtain this manual page:

```
nroff -man -TX file |
x9700 -h12 -v2 -l66 -TX -k prin1 1 -k prin2 2 -k prin3 3 \
-lk prin1 4 -lk prin2 5 -k vgraf 6 -k sdisc 7
```

To obtain viewgraphs:

```
nroff -TX - file <<eof |
.pl 35
.ll 45
eof
x9700 -s vint20 -TX
```

Chapter 3
TEXT FORMATTER'S INTRODUCTION

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

C3

- "Document Preparation Introduction" and "User Reference Manual"
- Macro Packages Reference: "Memorandum Macros User Guide"
  "Viewgraph Macros User Guide"
- Preprocessors Reference: "Table Formatting Program"
  "Pic Graphics Language"
  "Mathematics Typesetting Program"

The beginning user should refer to the Text Preparation System manual chapter entitled "User Reference Manual" for a better overall description of the text processing tools available on the UNIX system.

TEXT FORMATTERS

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

---

* Trademark of AT&T Bell Laboratories.

† Trademark of AT&T Technologies.

The text formatters covered in this book are:

- Chapter 4 – (*NROFF/TROFF TUTORIAL*) – Presents examples and explanations of formatting activities that you would use with **nroff** and **troff.**

- Chapter 5 – (*NROFF/TROFF USER MANUAL*) – Provides a description of **nroff**, a formatter designed to produce output for simple typewriter-like terminals, **troff**, a formatter designed to produce output for phototypesetters that is phototypesetter independent, and **otroff**, a formatter designed to produce output for the Wang Laboratories CAT phototypesetter only. The language of these three formatters is very similar and in most cases they are compatible.

- Chapter 6 – (*DEVICE INDEPENDENT TROFF*) – Provides a description of the **troff** formatter in which the differences between **troff** and **otroff** are pointed out.

Chapter 4

NROFF/TROFF TUTORIAL

OVERVIEW

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

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

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

These three programs are nroff/troff preprocessors. More information on the preprocessor can be found in the "Mathematics Typesetting Program", "Table Formatting Program", and "Pic Graphics Language" chapters of this manual.

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

troff formatter language. They should be considered first. More information on the macro packages is in the "Memorandum Macros User Guide" and the "Viewgraph Macros User Guide" chapters of this manual.

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

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

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

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

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

TUTORIAL TOPICS

1. Point Sizes and Line Spacing

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

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

Relative size changes are also legal and useful:

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

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

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

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

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

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

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

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

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

2. Fonts and Special Characters

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

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

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

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

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

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

bold*face* text

is produced by

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

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

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

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

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

```
.fp 3 H
```

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

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

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

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

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

3. Indents and Line Lengths

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

```
.ll 6i
```

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

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

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

will create a block of text that looks like:

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

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

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

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

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

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

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

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

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

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

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

4. Tabs

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

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

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

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

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

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

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

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

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

produces:

```
Name _________________Age ________
```

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

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

5. Local Motions

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

5.1 Vertical Motions

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

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

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

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

There are other ways to specify the amount of motion:

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

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

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

5.2 Horizontal Motions

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

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

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

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

to produce >>.

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

```
\w'thing'
```

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

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

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

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

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

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

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

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

which produces

```
systéme téléphonique
```

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

5.3 Overstrikes

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

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

5.4 Drawing Lines

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

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

6. Strings

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

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

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

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

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

```
téléphone
```

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

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

is the input.

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

```
.ds xx "        text
```

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

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

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

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

7. Introduction to Macros

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

8. Titles, Pages, and Page Numbering

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

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

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

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

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

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

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

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

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

These requests are explained as follows:

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

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

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

.bp	begin page
.br	break

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

9. Number Registers and Arithmetic

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

- Number registers hold only integers. In the **troff** formatter, arithmetic uses truncating integer division just like Fortran.
- In the absence of parentheses, evaluation is done left-to-right without any operator precedence, including relational operators. Thus:

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

becomes **-1**.

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

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

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

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

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

10. Macros With Arguments

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

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

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

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

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

The macro call would appear:

```
.SM SMALL
```

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

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

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

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

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

```
.SM ABLE ),
```

would appear (with "ABLE" in smaller type)

```
ABLE),
```

The macro call

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

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

```
(BAKER).
```

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

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

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

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

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

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

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

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

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

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

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

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

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

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

The fields could be made into parameters by using instead

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

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

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

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

11. Conditionals

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

12. Environments

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

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

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

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

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

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

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

13. Diversions

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

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

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

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

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

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

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

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

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

TUTORIAL EXAMPLES

Although the **nroff** and **troff** formatters have by design a syntax reminiscent of earlier text processors with the intent of easing their use, it is usually necessary to prepare at least a small set of macro definitions to describe most documents. Such common formatting needs as page margins and footnotes are deliberately not built into the **nroff** and **troff** formatters. Instead, the macro and string definition, number register, diversion, environment switching, page-position trap, and conditional input mechanisms provide the basis for user-defined implementations.

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

1. Page Margins

Header and footer macros are defined to describe the top and bottom page margin areas, respectively. A trap is planted at page position 0 for the header and at $-N$ (N from the page bottom) for the footer. A simple header and footer macro definition is

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

This example provides blank 1-inch top and bottom margins. The header will occur on the first page, only if the definition and trap exist prior to the initial pseudopage transition. In fill mode, the output line that springs the footer trap was typically forced out because some part or whole word did not fit on it. If anything in the footer and header that follows causes a break, that word or part word

will be forced out. In this and other examples, requests like **bp** and **sp** that normally cause breaks are invoked using the *no-break* control character (**'**). When the header/footer design contains material requiring independent text processing, the environment may be switched to avoid interaction with running text.

A more realistic example follows:

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

This example sets the size, font, and base-line spacing parameters for the footer material. Parameters are restored to their original values when the header is completed. The material in this case is a page number at the bottom of the first page and at the top of the remaining pages. If the **troff** formatter is used, a cut mark is drawn in the form of *root-en*'s at each margin. The **sp**'s refer to absolute positions to avoid dependence on the base-line spacing. Another reason for the **sp** in the footer is that the footer is invoked by printing a line whose vertical spacing swept past the trap position by possibly as much as the base-line spacing. The *no-space* mode is turned on at the end of **hd** to render ineffective accidental occurrences of **sp** at the top of the running text.

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

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

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

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

2. Paragraphs and Headings

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

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

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

A macro to automatically number section headings might look like:

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

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

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

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

The intended usage is

```
.lp label
```

The label will begin at 0.2 inch and cannot exceed a length of 0.3 inch without intruding into the paragraph. The label could be right adjusted against 0.4 inch by setting the tabs instead with

```
.ta 0.4iR 0.5i
```

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

3. Multiple Column Output

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

```
.de hd          \" header
---
.nr cl 0 1      \" initial column count
.mk             \" mark top of text
..
.de fo          \" footer
.ie \\n+(cl<2 \{\
.po +3.4i       \" next column; 3.1+0.3
.rt             \" back to mark
.ns \}          \" no-space mode
.el \{\
.po \\nMu       \" restore left margin
---
'bp \}
..
.ll 3.1i        \" column width
.nr M \\n(.o    \" save left margin
```

Typically, a portion of the top of the first page contains full width text; the request for the narrower line length, as well as another **.mk** request, will be made where the 2-column output is to begin.

4. Footnote Processing

The footnote mechanism is used by embedding the footnotes in the input text at the point of reference demarcated by an initial **.fn** and a terminal **.ef**.

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


The following macro definitions cause footnotes to be processed in a separate environment and diverted for later printing in the space immediately prior to the bottom margin. There is provision for the case where the last collected footnote does not completely fit in the available space:

```
.de hd                  \" header
---
.nr x 0 1               \" initial footnote count
.nr y 0-\\nb            \" current footer place
.ch fo -\\nbu           \" reset footer trap
.if \\n(dn .fz          \" leftover footnote
..
.de fo                  \" footer
.nr dn 0                \" zero last diversion size
.if \\nx \{\
.ev 1                   \" expand footnotes in environment 1
.nf                     \" retain vertical size
.FN                     \" footnotes
.rm FN                  \" delete it
.if '\\n(.z'fy' .di     \" end overflow diversion
.nr x 0                 \" disable fx
.ev     \}              \" pop environment
---
'bp
..
.de fx                  \" process footnote overflow
```

```
.if \\nx .di fy              \" divert overflow
..
.de fn                       \" start footnote
.da FN                       \" divert (append) footnote
.ev 1                        \" in environment 1
.if \\n+x=1 .fs              \" if first, include separator
.fi                          \" fill mode
..
.de ef                       \" end footnote
.br                          \" finish output
.nr z \\n(.v                 \" save spacing
.ev                          \" pop environment
.di                          \" end diversion
.nr y -\\n(dn                \" new footer position
.if \\nx=1 .nr y -(\\n(.v-\\nz)\  \" uncertainty correction
.ch fo \\nyu                 \" y is negative
.if (\\n(nl+1v)>(\\n(.p+\\ny)\
.ch fo \\n(nlu+1v            \" it did not fit
..
.de fs                       \" separator
\l'1i'                       \" 1 inch rule
.br
..
.de fz                       \" get leftover footnote
.fn
.nf                          \" retain vertical size
.fy                          \" where fx put it
.ef
..
.nr b 1.0i                   \" bottom margin size
.wh 0 hd                     \" header trap
.wh 12i fo                   \" footer trap, temp position
.wh -\\nbu fx                \" fx at footer position
.ch fo -\\nbu                \" conceal fx with fo
```

- The header macro (**hd**) initializes a footnote count register **x** and sets both the current footer trap position register **y** and the footer trap itself to a nominal position specified in register **b**.

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

- The footnote start macro (**fn**) begins a diversion (append) in environment 1 and increments the footnote count register **x**; if the count is one, the footnote separator macro (**fs**) is interpolated. The separator is kept in a separate macro to permit user redefinition.

- The footnote end macro (**ef**) restores the previous environment and ends the diversion after saving spacing size in register **z**.

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

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

- The footer trap is set to the lower of **y** or the current page position (**nl**) plus one line to allow for printing the reference line.

- If indicated by **x**, the footer **fo** rereads the footnotes from **FN** in no-fill mode in environment 1 and deletes **FN**. If the footnotes were too large to fit, the macro **fx** will be trap-invoked to redivert the overflow into **fy**, and the register **dn** will later indicate to the header whether or not **fy** is empty.

- Both **fo** and **fx** macros are planted in the nominal footer trap position in an order that causes **fx** to be concealed unless the **fo** trap is moved.

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

5. Last Page

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

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

During the eject, any traps encountered are processed normally. At the end of this last page, processing terminates unless a partial line, word, or partial word remains. If it is desired that another page be started, the end-macro will deposit a null partial word and effect another last page.

Chatper 5

NROFF AND TROFF USER MANUAL

1. Introduction

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

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

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

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

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

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

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

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

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

> *Note*: Throughout this chapter, a reference to **troff** also means **otroff** unless otherwise indicated. Where there are differences between the two, the differences are pointed out. Refer to Chapter 6 for a complete description of the device independent troff text formatter.

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

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

2. Usage

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

nroff *options files*
or
otroff *options files*
or
troff *options files*

where *options* represents any of a number of option arguments and *files* represents the list of files containing the document to be formatted. An argument consisting of a single minus sign (-) is taken to be a file name corresponding to the standard input. Input is taken from the standard input if no file names are given. Options may appear in any order so long as they appear before the files.

nroff, otroff AND troff OPTIONS

Option	*Effect*
−o*list*	Prints only pages whose page numbers appear in *list*, which consists of comma-separated numbers and number ranges. • A number range has the form *N*–*M* and means pages *N* through *M* • An initial –*N* means from the beginning to page *N* • A final *N*– means from page *N* to the end.
−n*N*	Number the first generated page *N*.
−s*N*	Stop every *N* pages. The **nroff** formatter will halt after every *N* pages (default *N*= 1) to allow paper loading or changing and will resume upon receipt of a new line. The **otroff** formatter will stop the phototypesetter every *N* pages, produce a trailer to allow changing cassettes, and resume after the phototypesetter is restarted. When using

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

−m*name* Prepend the macro file

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

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

−c*name* Prepend the compacted macro files

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

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

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

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

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

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

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

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

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

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

nroff ONLY OPTIONS

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

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

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

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

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

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

troff/otroff ONLY OPTIONS

Option	*Effect*

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

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

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

–w Wait until phototypesetter is available if busy.

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

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

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

---

* Registered trademark of Trendata Corporation.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

  tbl files | eqn -Tcat | otroff -t [options] | otc

  or

  tbl files | eqn | troff [options] | tc

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

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

* IMAGEN and IMPRINT-10 are registered trademarks of the IMAGEN Corporation.

3. NROFF/TROFF Reference Manual

3.1 General Explanation

3.1.1 Form of Input

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

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

3.1.2 Formatter and Device Resolution

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

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

3.1.3 Numerical Parameter Input

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

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

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

Figure 3-1. Nroff/Troff Scale Indicators

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

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

All other requests ignore scale indicators. When a number register containing an already appropriately scaled number is interpolated to provide numerical input, the basic unit scale indicator (**u**) may need to be appended to prevent an additional inappropriate default scaling. The number, *N*, may be specified in decimal-fraction form; but the parameter finally stored is rounded to an integer number of basic units.

The absolute position indicator (|) may be prepended to a number *N* to generate the distance to the vertical or horizontal place *N*.

- For vertically oriented requests and functions, |*N* becomes the distance in basic units from the current vertical place on the page or in a diversion {3.7} to the vertical place *N*.
- For all other requests and functions, |*N* becomes the distance from the current horizontal place on the input line to the horizontal place *N*.

For example:

```
.sp |3.2c
```

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

3.1.4 Numerical Expressions

Wherever numerical input is expected, an expression involving parentheses, the arithmetic operators

+, - , /, *, % (mod)

and the logical operators

<, >, <=, >=, = (or ==), & (and), : (or)

may be used. Except where controlled by parentheses, evaluation of expressions is left to right; there is no operator precedence. In the case of certain requests, an initial + or − is stripped and interpreted as an increment or decrement indicator. In the presence of default scaling, the desired scale indicator must be attached to every number in an expression for which the desired and default scaling differ. For example, if the number register **x** contains 2 and the current point size is 10, then:

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

will set the line length to ½ the sum of 4.25 inches + 2 picas + 3 ems (30 points since the point size is 10).

3.1.5 Notation

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

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

3.2 Font and Character Size Control

3.2.1 Fonts

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

Times Roman

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

Times Italic

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

Times Bold

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

Special Mathematical Font

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

Figure 3-2. Some Font Style Examples

The default fonts available with the **troff** formatter depend on the intended phototypesetter. Refer to "Device-Independent Troff" for font style examples for the AUTOLOGIC Incorporated, APS-5 phototypesetter.

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

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

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

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

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

3.2.2 Character Set

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

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

Figure 3-3. Troff ASCII Character Mapping

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

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

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

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

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

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

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

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

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

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

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

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

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

3.2.3 Character Size

For the **otroff** formatter, character point sizes available are 6, 7, 8, 9, 10, 11, 12, 14, 16, 18, 20, 22, 24, 28, and 36. This is a range of 1/12 inch to 1/2 inch. Character sizes with the **troff** formatter depend on the phototypesetter installation. The **.ps** request is used to change or restore the point size. Alternatively, the point size may be changed between any two characters by embedding a **\s***N* at the desired point to set the size to *N* or a **\s** $\pm N$ ($1 \leq N \leq 9$) to increment/decrement the size by *N*; **\s0** restores the previous size. Requested point size values that are between two valid sizes yield the larger of the two for **otroff**. With **troff**, requested point size values that are between two valid sizes yield the closer of the two. The current size is available in the **.s** number register. The **nroff** formatter ignores type size control. Part 8, *Character Size Control Requests*, contains a summary and explanation of character size requests.

3.3 Page Control

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

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

3.4 Text Filling, Adjusting, and Centering

3.4.1 Filling and Adjusting

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

An input text line ending with ., ?, :, or ! is taken to be the end of a sentence, and an additional space character is automatically provided during filling. Multiple interword space characters found in the input are retained, except for trailing spaces; initial spaces also cause a break.

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

A text input line that happens to begin with a control character can be made to not look like a control line by prefacing it with the nonprinting, zero-width filler character (**\&**). Another way is to specify output translation of some convenient character into the control character using the **.tr** request.

3.4.2 Interrupted Text

Copying of an input line in no-fill mode can be interrupted by terminating the partial line with a **\c** escape sequence. The next encountered input text line will be considered to be a continuation of the same line of input text. Similarly, a word within filled text may be interrupted by terminating the word (and line) with **\c**; the next encountered text will be taken as a continuation of the interrupted word. If the intervening control lines cause a break, any partial line will be forced out along with any partial word.

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

3.5 Vertical Spacing

3.5.1 Base-Line Spacing

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

3.5.2 Extra Line Space

If a word contains a vertically tall construct requiring the output line containing it to have extra vertical space before and/or after it, the *extra line space* function **\x'***N***'** can be embedded in or attached to that word. In this and other functions having a pair of delimiters around their parameter, the delimiter choice is arbitrary except that it cannot look like the continuation of a number expression for *N*.

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

- If *N* is positive, the output line containing the word will be followed by *N* extra vertical spaces.
- If successive requests for extra space apply to the same line, the maximum value is used.

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

3.5.3 Blocks of Vertical Space

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

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

3.6 Line Length and Indenting

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

3.7 Macros, Strings, Diversions, and Position Traps

3.7.1 Macros and Strings

A macro is a named set of arbitrary lines that may be invoked by name or with a trap. A string is a named string of characters, not including a newline character, that may be interpolated by name at any point. Request, macro, and string names share the same name list. Macro and string names may be 1- or 2-characters long and may usurp previously defined request, macro, or string names. Any of these entities may be renamed with **.rn** or removed with **.rm**.

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

A macro is invoked in the same way as a request; a control line beginning **.xx** will interpolate the contents of macro **xx**. The remainder of the line may contain up to nine arguments. The strings **x** and **xx** are interpolated at any desired point with *x and *(xx, respectively. String references and macro invocations may be nested.

3.7.2 Copy Mode Input Interpretation

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

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

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

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

3.7.3 Arguments

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

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

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

and called by

```
.xx Monday 14th
```

to produce the text

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

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

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

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

3.7.4 Diversions

Processed output may be diverted into a macro for purposes such as footnote processing or determining the horizontal and vertical size of some text for conditional changing of pages or columns. A single diversion trap may be set at a specified vertical position. The number registers **.dn** and **.dl**, respectively, contain the vertical and horizontal size of the most recently ended diversion. Processed text that is diverted into a macro retains the vertical size of each of its lines when reread in no-fill mode regardless of the current *V*. Constant-spaced (**.cs**) or emboldened (**.bd**) text that is diverted can be reread correctly only if these modes are again or still in effect at reread time. One way to do this is to embed in the diversion the appropriate **.cs** or **.bd** request with the transparent mechanism described in paragraph 3.10.6.

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

- Diversion trap and associated macro

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

3.7.5 Traps

Three types of trap mechanisms are available:

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

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

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

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

3.8 Number Registers

A variety of predefined number registers (Part 5) are available to the user. In addition, the user may define his own named registers. Register names are 1- or 2-characters long and do not conflict with request, macro, or string names. Except for certain predefined read-only number registers (Part 6), a number register can be read, written, automatically incremented or decremented, and interpolated into the input in a variety of formats. One common use of user-defined registers is to automatically number sections, paragraphs, lines, etc. A number register may be used any time numerical input is expected or desired and may be used in numerical expressions.

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

SEQUENCE	*EFFECT ON REGISTER*	*VALUE INTERPOLATED*
\n*x*	none	*N*
\n(*xx*	none	*N*
\n+*x*	*x* incremented by *M*	*N+M*
\n-*x*	*x* decremented by *M*	*N-M*
\n+(*xx*	*xx* incremented by *M*	*N+M*
\n-(*xx*	*xx* decremented by *M*	*N-M*

Figure 3-6. Nroff/Troff Number Register Interpolation

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

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

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

3.9 Tabs, Leaders, and Fields

3.9.1 Tabs and Leaders

The ASCII horizontal tab character and the ASCII SOH character (the leader) can both be used to generate either horizontal motion or a string of repeated characters. The length of the generated entity is governed by internal tab stops specified with a **.ta** request. The default difference is that tabs generate motion and leaders generate a string of periods; **.tc** and **.lc** offer the choice of repeated character or motion. There are three types of internal tab stops: left justified, right justified, and centered. In Figure 3-7:

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

TAB TYPE	*LENGTH OF MOTION OR REPEATED CHARACTERS*	*LOCATION OF next-string*
Left	*D*	Following *D*
Right	*D-W*	Right justified within *D*
Centered	*D-W/2*	Centered on right end of *D*

Figure 3-7. Nroff/Troff Tab Types

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

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

3.9.2 Fields

A field is contained between a pair of field delimiter characters. It consists of substrings separated by padding indicator characters. The field length is the distance on the input line from the position where the field begins to the next tab stop. The difference between the total length of all the substrings and the field length is incorporated as horizontal padding space that is divided among the indicated padding places. The incorporated padding is allowed to be negative. For example, if the field delimiter is "#" and the padding indicator is "^", then

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

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

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

3.10 Input/Output Conventions and Character Translations

3.10.1 Input Character Translations

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

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

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

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

3.10.2 Ligatures

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

3.10.3 Backspacing, Underlining, and Overstriking

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

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

3.10.4 Control Characters

Both the *break* control character (.) and the *no-break* control character (') may be changed, if desired. Such a change must be compatible with the design of any macros used in the span of the change and particularly of any trap-invoked macros. A summary and explanation of the **.cc** and **.c2** control character requests are included in Part 16.

3.10.5 Output Translation

One character can be made a stand-in for another character using the **.tr** request. All text processing (e.g., character comparisons) takes place with the input (stand-in) character which appears to have the width of the final character. Graphic translation occurs at the moment of output (including diversion). Included in Part 16 is a summary and explanation of the output translation request.

3.10.6 Transparent Throughput

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

3.10.7 Comments and Concealed Newline Characters

An uncomfortably long input line that must stay on one line (e.g., a string definition or no-filled text) can be split into many physical lines by ending all but the last one with the escape character (\). The sequence **\<newline>** is ignored except in a comment. Comments may be embedded at the end of any line by prefacing them with \". The newline character at the end of a comment cannot be concealed. A line beginning with \" will appear as a blank line and

behave like **.sp 1**; a comment can be on a line by itself by beginning the line with .\".

3.11 Local Horizontal/Vertical Motion and Width Function

3.11.1 Local Motion

The functions \v'*N*' and \h'*N*' can be used for local vertical and horizontal motion, respectively. The distance *N* may be negative; the positive directions are *rightward* and *downward*. A local motion is one contained within a line. To avoid unexpected vertical dislocations, it is necessary that the net vertical local motion (within a word in filled text and otherwise within a line) balance to zero. The above and certain other escape sequences providing local motion are summarized and explained in Figure 3-8 and Figure 3-9. As an example, E^2 is generated by the sequence **E\v'-.5'\s-4\&2\s0\v'.5'**.

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

Figure 3-8. Vertical Local Motions

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

Figure 3-9. Horizontal Local Motions

3.11.2 Width Function

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

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

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

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

- **0** means that all characters in *string* are short lowercase characters without descenders (like **e**).
- **1** means that at least one character has a descender (like **y**).
- **2** means that at least one character is tall (like **H**).
- **3** means that both tall characters and characters with descenders are present.

3.11.3 Mark Horizontal Place

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

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

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

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

3.12.1 Overstrike

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

3.12.2 Zero-Width Characters

The function \z*c* will output *c* without spacing over it and can be used to produce left-aligned overstruck combinations. As examples, "\z\(ci\(pl" will produce ⊕, and "\(br\z\(rn\(ul\(br" will produce the smallest possible constructed box (□).

3.12.3 Large Brackets

The Special Mathematical Font contains a number of bracket construction pieces that can be combined into various bracket styles. The function \b'*string*' may be used to pile up vertically the characters in *string* (the first character on top and the last at the bottom); the characters are vertically separated by one em and the total pile is centered one-half em above the current base line (one-half line in the **nroff** formatter). For example:

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

produces ⎡E⎤.

3.12.4 Line Drawing

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

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

- If *c* is not specified, the base-line rule (_) is used (underline character in **nroff**).
- If *N* is negative, a backward horizontal motion of size *N* is made before drawing the string.

Any space resulting from *N*/(size of *c*) having a remainder is put at the beginning (left end) of the string. In the case of characters that are designed to be connected, such as base-line rule (_), underrule (\(**ul**), and root en (\(**ru**), the remainder space is covered by overlapping. If *N* is less than the width of *c*, a single *c* is centered on a distance *N*. As an example, a macro to underscore a string can be written:

```
.de us
\\$1\l'¦0\(ul'
..
```

or one to draw a box around a string:

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

such that

.us " underlined words"

and

.bx " words in a box"

yield

underlined words

and

| words in a box |

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

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

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

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

3.13 Hyphenation

The automatic hyphenation may be switched off and on. When switched on with **.hy**, several variants may be set. A hyphenation indicator character may be embedded in a word to specify desired hyphenation points or may be prepended to suppress hyphenation. In addition, the user may specify a small exception word list. The default condition of hyphenation is off.

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

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

3.14 Three-Part Titles

The titling function **.tl** provides for automatic placement of three fields at the left, center, and right of a line with a title length specifiable with **.lt**. The **.tl** may be used anywhere and is independent of the normal text collecting process. A common use is in header and footer macros. Part 18 is a summary and explanation of 3-part title requests.

3.15 Output Line Numbering

Automatic sequence numbering of output lines may be requested with **.nm**. When in effect, a 3-digit, Arabic number plus a digit space is prepended to output text lines. Text lines are offset by four digit spaces and otherwise retain their line length. A reduction in line length may be desired to keep the right margin aligned with an earlier margin. Blank lines, other vertical spaces, and lines generated by **.tl** are not numbered. Numbering can be temporarily suspended with **.nn** or with a **.nm** followed by a later **.nm +0**. In addition, a line number indent *I* and the number-text separation *S* may be specified in digit spaces. Further, it can be specified that only those line numbers that are multiples of some number *M* are to be printed (the others will appear as blank number fields). Part 19 is a summary and explanation of output line numbering requests.

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

- **.nm 1 3** was placed at the beginning.
- **.nm +0** was placed in front of the second and third paragraphs.
- **.nm** was placed at the end.

Line lengths were also changed (by **\w'0000'u**) to keep the right side aligned. Another example is:

```
.nm +5 5 x 3
```

which turns on numbering with the line number of the next line to be five greater than the last numbered line, with M=5, spacing S untouched, and the indent I set to 3.

Automatic sequence numbering of output lines may be requested with **.nm**. When in effect, a 3-digit, arabic number plus a digit-space is prepended to output text lines. Text lines are offset by four digit-spaces and otherwise retain their line length. A reduction in line length may be desired to keep the right margin aligned with an earlier margin. Blank lines, other vertical spaces, and lines generated by **.tl** are not numbered. Numbering can be temporarily suspended with **.nn** or with a **.nm** followed by a later **.nm +0**. In addition, a line number indent *I* and the number-text separation *S* may be specified in digit-spaces. Further, it can be specified that only those line numbers that are multiples of some number *M* are to be printed (the others will appear as blank number fields). Part 19 is a summary and explanation of output line numbering requests.

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

Figure 3-10. Example of Output Line Numbering

3.16 Conditional Acceptance of Input

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

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

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

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

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

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

Any spaces between the condition and the beginning of *anything* are skipped over. The *anything* can be either a single input line (text, macro, or whatever) or a number of input lines. In the multiline case, the first line must begin with a left delimiter "\{", and the last line must end with a right delimiter "\}". If the left delimiter is the last thing on that line, the following newline should be concealed with a "\" or a blank line may result on output.

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

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

outputs a title if the page number is even, and

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

treats Page 1 differently from other pages.

3.17 Environment Switching

A number of parameters that control text processing are gathered together into an environment, that can be switched by the user. Environment parameters are those associated with some requests. Parts 7 through 25 of this section indicate in the "Explanation" column those requests so affected. In addition, partially collected lines and words are in the environment. Everything else is global; examples are page-oriented parameters, diversion-oriented parameters, number registers, and macro and string definitions. All environments are initialized with default parameter values. Part 21 is a summary and explanation of the environment switching request.

3.18 Insertions From Standard Input

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

If insertions are to be taken from the terminal keyboard while output is being printed on the terminal, the command line option **−q** will

turn off the echoing of keyboard input and prompt only with BEL. The regular input and insertion input cannot simultaneously come from the standard input. As an example, multiple copies of a form letter may be prepared by entering insertions for all copies in one file to be used as the standard input and causing the file containing the letter to reinvoke itself by using the **.nx** request. The process would be ended by a **.ex** request in the insertion file. Part 22 is a summary and explanation of insertions from the standard input requests.

3.19 Input/Output File Switching

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

3.20 Miscellaneous

Part 24 is a summary and explanation of miscellaneous requests.

3.21 Output and Error Messages

Output from **.tm**, **.pm**, and prompt from **.rd**, as well as various error messages are written onto the UNIX system standard output and error (message) output. By default, both are written onto the user's terminal, but they can be independently redirected.

Various error conditions may occur during the operation of the **nroff** and **troff** formatters. Certain less serious errors having only local impact do not cause processing to terminate. Two examples are:

- *word overflow*—caused by a word that is too large to fit into the word buffer (in fill mode).
- *line overflow*—caused by an output line that grew too large to fit in the line buffer.

In both cases, a message is printed, the offending excess is discarded, and the affected word or line is marked at the point of truncation with an * (in **nroff**) or a ☞ (in **troff**). The philosophy is to continue processing, if possible, on the grounds that output useful for debugging may be produced. If a serious error occurs, processing

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

3.22 Compacted Macros

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

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

3.22.1 Building a Compacted Macro Package

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

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

Compactible Material

.
.
.

.co
Noncompactible Material

.
.
.

3.22.2 Produce Compacted Files

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

nroff -k*mac mac*

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

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

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

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

---

* Trademark of Digital Equipment Corporation.

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

3.22.3 Install Compacted Files

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

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

C-5

3.22.4 Install Noncompactible Segment

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

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

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

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

4. Nroff/Troff Escape Sequences

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

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

C-5

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

5. Predefined General Number Registers

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

6. Predefined Read-Only Number Registers

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

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

7. Font Control Requests

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

Embolden font *F* by *N*-1 units. Characters in font *F* will be artificially emboldened by printing each one twice, separated by *N*-1 basic units. A reasonable value for *N* is 3 when the character size is in the vicinity of 10 points. If *N* is missing, the embolden mode is turned off. The mode must still (or again) be in effect when the characters are physically printed. There is no effect in the **nroff** formatter.

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

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

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

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

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

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

8. Character Size Control Requests

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

Set constant character space (width) mode on for font *F* (if mounted). The width of every character is assumed to be *N*/36 ems. If *M* is absent, the em is that of the character point size; if *M* is given, the em is *M*-points. All affected characters are centered in this space including those with an actual width larger than this space. Special font characters occurring while the current font is *F* are also so treated. If *N* is absent, the mode is turned off. The mode must still (or again) be in effect when the characters are printed. There is no effect in the **nroff** formatter.

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

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

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

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

9. Page Control Requests

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

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

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

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

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

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

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

- If the distance to the next trap position (D) is less than N, a forward vertical space of size D occurs which will spring the trap.
- If there are no remaining traps on the page, D is the distance to the bottom of the page.
- If D is less than vertical spacing (V), another line could still be output and spring the trap.

In a diversion, D is the distance to the diversion trap (if any) or is very large. Mode or relevant parameters are associated with current diversion level.

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

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

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

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

C.5

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

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

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

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

10. Text Filling, Adjusting, and Centering Requests

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

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

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

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

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

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

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

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

Center. The next *N* input text lines are centered within the current line-length. If *N* = 0, any residual count is cleared. A break occurs after each of the *N* input lines. If the input line is too long, it will be left adjusted. The request normally causes a break. Relevant parameters are a part of the current environment.

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

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

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

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

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

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

11. Vertical Spacing Requests

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

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

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

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

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

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

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

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

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

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

Space vertically. The request provides spaces in either direction. If N is negative, the motion is backward (upward) and is limited to the distance to the top of the page. Forward (downward) motion is truncated to the distance to the nearest trap. If the no-space mode (**.ns**) is on, no spacing occurs. The scale indicator is ignored if not specified in the request. The request causes a break.

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

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

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

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

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

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

12. Line Length and Indenting Requests

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

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

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

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

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

Temporary indent. The next output text line will be indented a distance $\pm N$ with respect to the current indent. The resulting total indent may not be negative. The current indent is not changed. The scale indicator is ignored if not specified in the request. Relevant parameters are a part of the current environment. The request causes a break.

13. Macro, String, Diversion, and Trap Requests

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Install a diversion trap at position *N* in the current diversion to invoke macro *xx*. Another **.dt** will redefine the diversion trap. If no arguments are given, the diversion trap is removed. Mode or relevant parameters are associated with current diversion level. The scale indicator is ignored if not specified in the request.

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

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

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

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

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

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

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

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

REQUEST FORM	*INITIAL VALUE*	*IF NO ARGUMENT*
.wh *Nxx*	-	-

When. A location trap is set to invoke macro *xx* at page position *N*; a negative *N* is interpreted with respect to the page bottom. Any macro previously planted at *N* is replaced by *xx*. A zero *N* refers to the top of a page. In the absence of *xx*, the first found trap at *N*, if any, is removed. The scale indicator is ignored if not specified in the request.

14. Number Registers Requests

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

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

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

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

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

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

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

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

15. Tab, Leader, and Field Requests

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

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

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

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

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

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

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

t	*ADJUSTMENT TYPE*
R	right
C	centering
absent	left

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

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

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

16. Input/Output and Translation Requests

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

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

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

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

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

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

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

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

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

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

Turn escape character mechanism off.

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

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

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

Translate *a* into *b*, *c* into *d*, etc. on output. If an odd number of characters is given, the last one will be mapped into the space character. To be consistent, a particular translation must stay in effect from input to output time. Initially there are no translate values.

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

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

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

Underline in the **nroff** formatter (italicize in **troff**) the next *N* input text lines. Switch to underline font saving the current font for later restoration; other font changes within the span of a **.ul** will take effect, but the restoration will undo the last change. Output generated by **.tl** is affected by the font change but does not decrement *N*. If *N* is greater than 1, there is the risk that a trap interpolated macro may provide text lines within the span, which environment switching can prevent. Relevant parameters are a part of the current environment.

17. Hyphenation Requests

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

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

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

Exception words. Hyphenation points in words are specified with embedded minus signs. Versions of a word with terminal s are implied; i.e., *dig-it* implies *dig-its*. This list is examined initially and after each suffix stripping. Space available is small — about 128 characters.

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

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

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

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

18. Three-Part Title Requests

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

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

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

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

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

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

C-5

19. Output Line Numbering Requests

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

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

REQUEST FORM	*INITIAL VALUE*	*IFNO ARGUMENT*
.nn N	-	$N = 1$

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

20. Conditional Acceptance Requests

REQUEST FORM	*INITIAL VALUE*	*IF NO ARGUMENT*

.el *anything*

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

REQUEST FORM	*INITIAL VALUE*	*IF NO ARGUMENT*

.ie *c anything*

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

REQUEST FORM	*INITIAL VALUE*	*IF NO ARGUMENT*

.if *c anything*

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

REQUEST FORM	*INITIAL VALUE*	*IF NO ARGUMENT*

.if *!c anything*

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

REQUEST FORM	*INITIAL VALUE*	*IF NO ARGUMENT*

.if *N anything*

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

REQUEST FORM	*INITIAL VALUE*	*IF NO ARGUMENT*

.if *!N anything*

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

REQUEST FORM	*INITIAL VALUE*	*IFNO ARGUMENT*

.if *'s1' s2' anything*

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

REQUEST FORM	*INITIAL VALUE*	*IFNO ARGUMENT*

.if *!'s1' s2' anything*

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

21. Environment Switching Request

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

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

22. Insertions From Standard Input Requests

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

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

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

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

23. Input/Output File Switching Requests

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

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

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

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

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

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

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

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

24. Miscellaneous Requests

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

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

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

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

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

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

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

Sets margin character *c* and separation *N*. Specifies that a margin character *c* appear a distance *N* to the right of the right margin after each nonempty text line (except those produced by **.tl**). If the output line is too long (as can happen in no-fill mode), the character will be appended to the line. If *N* is not given, the previous *N* is used; the initial *N* is 0.2 inches in the **nroff** formatter and 1 em in **troff**. Relevant parameters are a part of the current environment. The scale indicator is ignored if not specified in the request.

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

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

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

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

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

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

25. Output and Error Messages Request

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

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

Chapter 6

DEVICE-INDEPENDENT TROFF

1. Introduction

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

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

2. Incompatibilities

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

2.1 Unsupported Features

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

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

```
.ft CW
.br
```

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

```
.br
.ft
```

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

2.2 Deleted Options

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

2.3 Deleted Requests

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

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

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

2.4 Deleted Escape Sequences and Number Registers

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

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

2.5 Changed Character Names in The CW Font

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

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

3. New Features

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

3.1 New Options

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

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

3.2 New Requests

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

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

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

3.3 Modified Requests and Escape Sequences

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

3.4 New Escape Sequences

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

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

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

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

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

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

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

3.5 New Predefined Number Registers and Strings

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

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

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

3.6 New Special Character Names

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

4. Other Important Changes

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

5. Caveats

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

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

6. Supported Devices

6.1 Phototypesetters

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

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

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

6.2 Simulator Postprocessors

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

7. Parameters of the APS-5 Phototypesetter

RESOLUTION: 723 units per inch.

LEGAL POINT SIZES:

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

DEFAULT FONT SETTINGS:

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

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

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

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

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

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

Chapter 7
MACROS 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. Collections of chapters that describe other aspects of the DOCUMENTER'S WORKBENCH software are:

- "Document Preparation Introduction" and "User Reference Manual"
- Text Formatters Reference: "Nroff/troff Tutorial"
  "Nroff and Troff User Manual"
  "Device Independent Troff"
- Preprocessors Reference: "Table Formatting Program"
  "Pic Graphics Language"
  "Mathematics Typesetting Program"

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 "User Reference Manual" for a better overall description of the text processing tools available.

C-7

Manual pages for the UNIX system commands are documented in the Runtime System manual and Software Development System manual. 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.

---

* 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 chapters on text formatters: "Nroff and Troff User Manual" and "Device-Independent Troff"

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:** (following) — 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:** (omitted) — 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:** (following) — 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. See "Viewgraph Macros User Guide."

C-7

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 8

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. Runtime System Manual section 1— **300, 4014, 450, col, ed, hp, sh,** and **spell**; Section 7 — **termio**(7).

2. Software Development System manual— **term**(4).

3. "User 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.

C-8

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).

C-8

* 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
```

C-8

- 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 24 of this part, and **300, 450, 4014, hp, col** in section 1 of the Runtime System manual and in **term, terminfo** of the Software Development 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 **t**bl 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.

C-8

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.

C-8

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.

C-8

- 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.

C-8

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

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.

C-8

- 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.

C-8

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

C-8

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.

C-8

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."*

C-8

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.

C-8

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 {1.8}.

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

C-8

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⒯center⒯right⒯left⒯center⒯right
⒯first column⒯⒯⒯second column
.fi
.sp 2
..
```

where ⒯ stands for the tab character.

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

12.7 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]

C-5

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 -s1 option on a document that is longer than ten pages.

Cannot execute *filename*

Given by the .! request if the *filename* is not found.

Cannot open *filename*

Indicates one of the files in the list of files to be processed cannot be opened.

Exception word list full

Indicates too many words have been specified in the hyphenation exception list (via **.hw** requests).

Line overflow

Indicates output line being generated was too long for the formatter line buffer capacity. The excess was discarded. Likely causes for this message are very long lines or words generated through the misuse of \c of the **.cu** request, or very long equations produced by **eqn/neqn**.

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 9

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. Runtime System Manual — **ed** and **spell.**

2. "User 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}.

C-9

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.
System⓽Bits/Byte⓽Bytes/Word⓽Bits/Word
_
IBM 7090/94⓽6⓽6⓽36
IBM 360/370⓽8⓽4⓽32
PDP 11/70⓽8⓽2⓽16
.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(T)Hardware
(T)_(T)_(T)_
(T)UNIX\*(Tm(T)Model(T)Serial
(T)System(T)\^(T)Number
=
OS Dev.(T)A(T)VAX(T)54
SGS Dev.(T)B(T)11/70(T)3275
Low-End(T)C(T)11/23(T)221

_
And now ...(T){
.NA
Some filled text and an equation:
T}(T)T{
$ zeta (s) = prod
from k=1 to inf k sup -s $
.AD
T}(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 Nroff/troff chapters 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.

C.9

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.

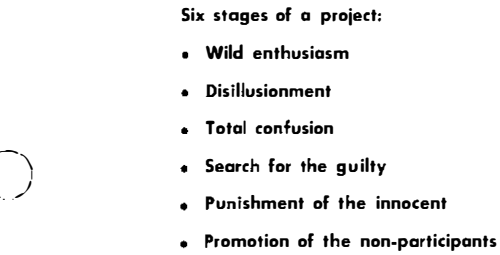
The "*(Tm" string generates a trademark symbol.

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.



C-9

Figure 4-1. Trivial Example (Not Actual Size)

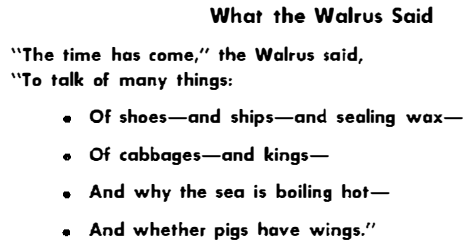


Figure 4-2. Less Trivial Example (Not Actual Size)

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)

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)

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.

C-9

Figure 4-5. Example of Indent (Not Actual Size)

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 — (NOTE 3) —	H	AR	1/AR	W — (NOTE 3) —	H	AR	1/AR
.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

C.9

NOTES

Chapter 10
PREPROCESSORS 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. Collections of chapters that describe other aspects of the DOCUMENTER'S WORKBENCH SOFTWARE ARE:

- "Document Preparation Introduction" and "User Reference Manual"
- Text Formatters Reference: "Nroff/troff Tutorial"
"Nroff and Troff User Manual"
"Device Independent Troff"
- Macro Packages Reference: "Memorandum Macros User Guide"
"Viewgraph Macros User Guide"

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 Text Preparation System manual chapter entitled "User 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.

C-10

---

* 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 manual.

- Chapter 11 – *Table Formatting Program* (tbl)
- Chapter 12 – *Picture Graphics Language* (pic)
- Chapter 13 – *Mathematics Typesetting Prgoram* (eqn)

Chapter 11

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.
Language⊤Authors⊤Runs on
.sp
_
Fortran⊤Many⊤Almost anything
C⊤BTL⊤11/45,H6000,370
BLISS⊤Carnegie-Mellon⊤PDP-10,11
IDS⊤Honeywell⊤H6000
Pascal⊤Stanford⊤370
.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
Year⓽Price⓽Dividend
1971⓽41-54⓽$2.60
2⓽41-54⓽2.70
3⓽46-55⓽2.87
4⓽40-53⓽3.24
5⓽45-52⓽3.40
6⓽51-59⓽.95*
.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
_
Bridge⓽Designer⓽Length
_
Brooklyn⓽J. A. Roebling⓽1595
Manhattan⓽G. Lindenthal⓽1470
Williamsburg⓽L. L. Buck⓽1600
_
Queensborough⓽Palmer &⓽1182
⓽ Hornbostel
_
⓽⓽1380
Triborough⓽O. H. Ammann⓽_
⓽⓽383
_
Bronx Whitestone⓽O. H. Ammann⓽2300
Throgs Neck⓽O. H. Ammann⓽1800
.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 12

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

C-12

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**

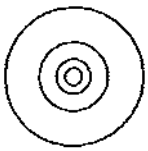
---

* 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

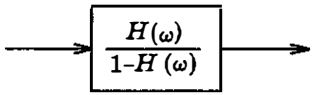
```
ellipse "\s8\fISmile!\fP\s0"
```

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**:

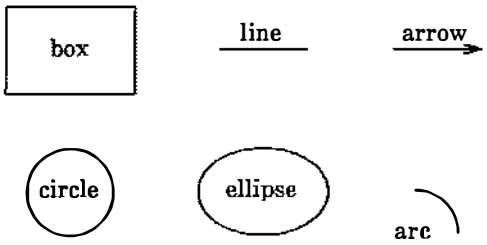
```
arrow
box "$space 0 {H( omega )} over {1 - H( omega )}$"
arrow
```



2. PIC User Manual

2.1 Basics

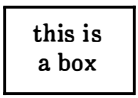
Pic provides boxes, lines, arrows, circles, ellipses, arcs, and splines (arbitrary smooth curves), plus facilities for positioning and labeling them. The picture below shows all of the fundamental objects (except for splines) in their default sizes:



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:

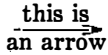
```
.de PS
.if t .sp .3
.in (\\n(.lu-\\$2u)/2u
.ne \\$1u
..
.de PE
.in
.if t .sp .6
..
```

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
```

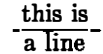
produces



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,

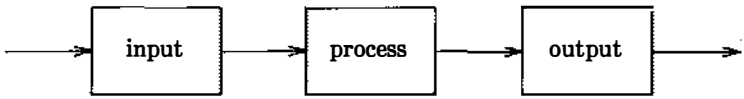
```
arc; arc; arc; arc
```

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
```

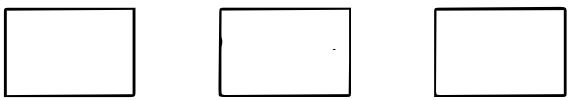
produces the figure



If you want to leave a space at some place, use `move`:

```
box; move; box; move; box
```

produces

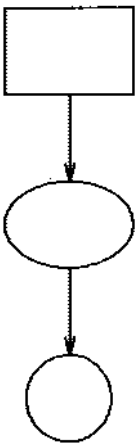


Notice that several commands can be put on a single line if they are separated by semicolons.

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

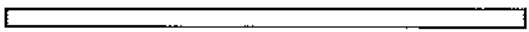
to choose sensible default values for these dimensions, so that simple pictures can be drawn with a minimum of fuss and bother. All of the figures and motions shown so far have been in their default sizes.

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.

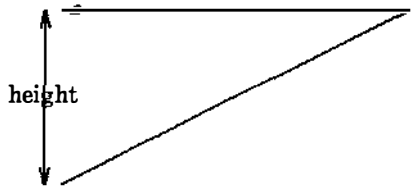
Giving an attribute like **`width`** changes only the one instance of the object. You can also change the default size for all objects of a particular type, as discussed later.

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

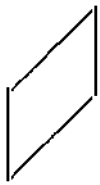


The notation `<->` indicates a two-headed arrow; use `->` for a head on the end and `<-` for one on the start. Lines and arrows are really the same thing; in fact, `arrow` is a synonym for `line ->`.

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



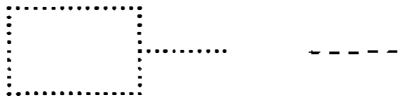
Warning: a very common error is to say

```
line 3i
```

A direction is needed.

```
line right 3i
```

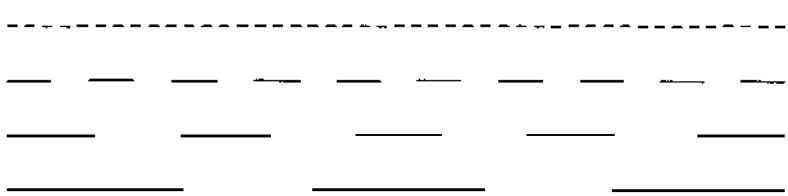
Boxes and lines may be dotted or dashed:



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

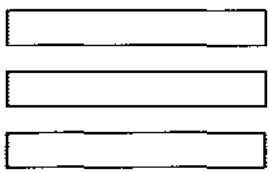
The "width" of an arrowhead is the distance across its tail; the "height" is the distance along the shaft. The arrowheads in this picture are default size.

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.

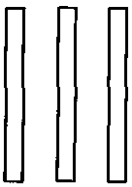
It is possible to change the default sizes of objects by assigning values to certain variables:

```
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.

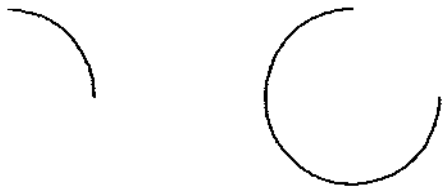
The `from` and `to` attributes are particularly useful with arcs, to specify the endpoints. By default, arcs are drawn counterclockwise,

```
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:

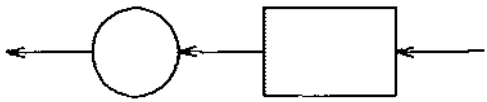
```
arc -> cw from 0,0 to 2i,0 rad 15i
```

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.

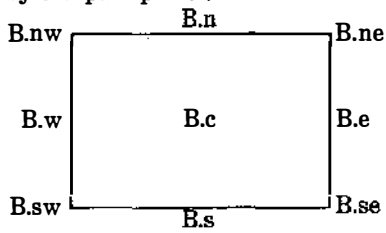
You can also refer to previously drawn objects of each type, using the word `last`. For example, given the input

```
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

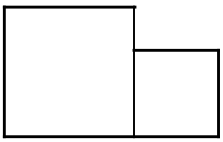
```
line from upper left of 2nd last box to bottom of 3rd last ellipse
```

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

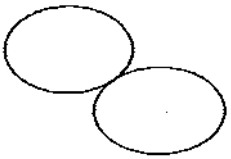


Notice that the corner after `with` is written `.sw`.

As another example, consider

```
ellipse; ellipse with .nw at last ellipse.se
```

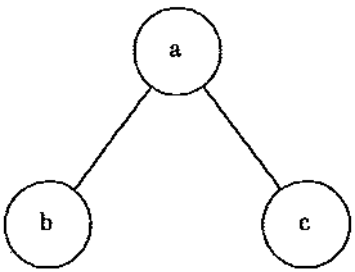
which makes



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*

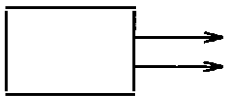
fraction is any expression, and *position1* and *position2* are any positions. You can abbreviate this phrase; "of the way" is optional, and the whole thing can be written instead as

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:

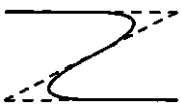
This line was produced by

```
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
```

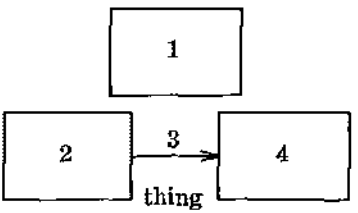
At the moment, arrowheads may only be put on the ends of a line or spline; splines may not be dotted or dashed.

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: ...;  ]
```

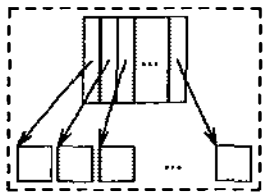
When combined with `define` statements (next section), blocks provide a reasonable simulation of a procedure mechanism.

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
```

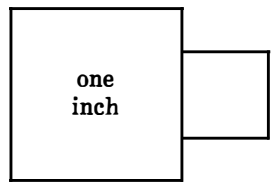
Then

```
square(1i, "one" "inch")
```

calls for a one-inch square with the obvious label, and

```
square(0.5i)
```

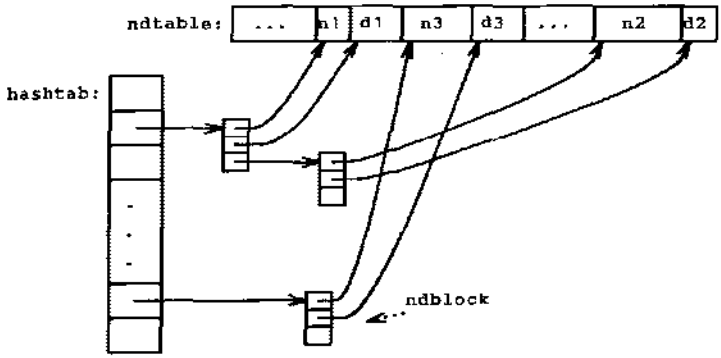
calls for a square with no label:



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

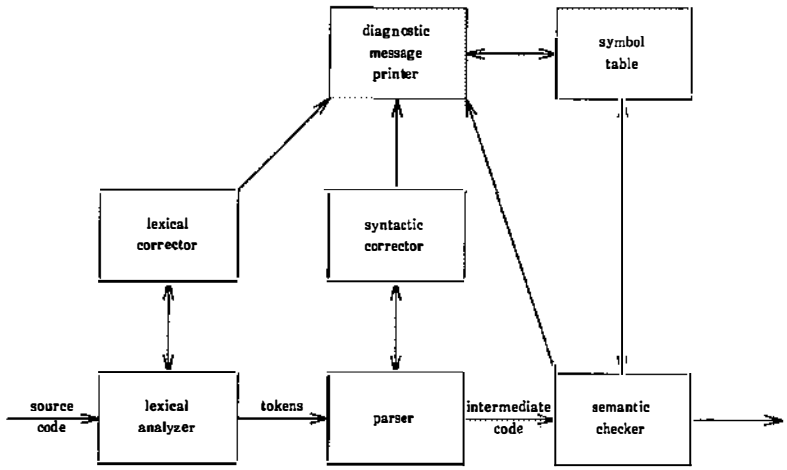
Here are a few larger examples:



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.



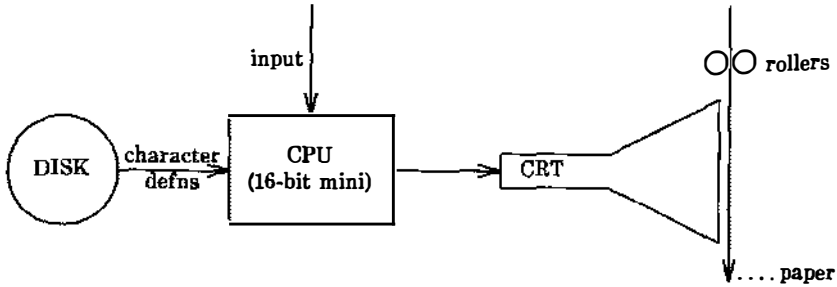
This input will generate a picture something like the above:

```
.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:
```
    nth
    nth 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 13

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 · 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

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

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

$$\int \quad \prod \quad \bigcup \quad \bigcap$$

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

xy

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

bold {x y} z

produces

$\mathbf{xy}z$

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

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

A = fat {pi r sup 2}

produces

$A = \pi r^2$

Legal sizes which may follow *size* are

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

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

size +2

makes the size two points larger. This has the advantage that

knowledge of the current size is not necessary.

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

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

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

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

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:

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

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

C-13

```
{ 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

$$\begin{matrix} x_i & x^2 \\ y_i & y^2 \end{matrix}$$

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.

NOTES