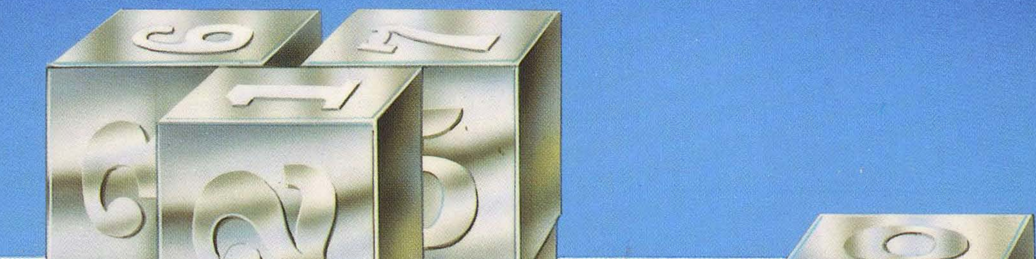
VOLUME 4

DOCUMENTATION PREPARATION



programmer's manual

CBS COLLEGE PUBLISHING'S
UNIX* SYSTEM LIBRARY

AT&T

* TRADEMARK OF AT&T

AT&T

VOLUME 4

PREPARATION DOCUMENTATION

UNIX* programmer's manual

CBS COLLEGE PUBLISHING'S
UNIX SYSTEM LIBRARY

* Trademark of AT&T.

AT&T

VOLUME 4

PREPARATION DOCUMENTATION

UNIX*

programmer's manual

CBS COLLEGE PUBLISHING'S
UNIX SYSTEM LIBRARY

Steven V. Earhart: Editor

HOLT, RINEHART AND WINSTON
New York   Chicago   San Francisco   Philadelphia
Montreal   Toronto   London   Sydney   Tokyo
Mexico City   Rio de Janeiro   Madrid

* Trademark of AT&T.

This document was set on an AUTOLOGIC, Inc. APS-5 phototypesetter driven by the TROFF formatter operating under the UNIX system on an AT&T 3B20 computer.

* Trademark of AT&T.

Library of Congress Cataloging-in-Publication Data

UNIX programmer's manual.

At head of title: AT&T
Inclues index.
Contents: v. 1. Commands and utilities—v. 2.
System calls and library routines—v. 3. System
administration facilities.—v. 4. Document Preparation.
1. UNIX (Computer operating system) I. Earhart,
Steven V. II. American Telephone and Telegraph Company.
QA76.76.063U548 1986 005.4′3 86-311

Select Code 320-034
ISBN 0-03-011207-9

Printed in the United States of America

Published simultaneously in Canada

6 7 8 090 9 8 7 6 5 4 3 2

CBS COLLEGE PUBLISHING
Holt, Rinehart and Winston
The Dryden Press
Saunders College Publishing

PREFACE

The *UNIX Programmer's Manual* describes most of the features of the UNIX System V. It does not provide a general overview of the UNIX system nor details of the implementation of the system.

Not all commands, features, or facilities described in this series are available in every UNIX system implementation. For specific questions on a machine implementation of the UNIX system, consult your system administrator.

The *UNIX Programmer's Manual* is available in several volumes.

- Volume 1 contains the Commands and Utilities (sections 1 and 6)
- Volume 2 contains the System Calls and Library Routines (sections 2,3,4, and 5).
- Volume 3 contains the System Administration Facilities (sections 1M, 7, and 8).
- Volume 4 contains the Document Preparation Facilities (**mm**, **tbl**, etc.).
- Volume 5 contains the Languages and Support Tools (C **language**, **lex**, **make**, etc.).

INTRODUCTION

An important feature of the UNIX* operating system is to provide a method of document preparation and generation. This manual provides information needed to make optimum use of the system. The following sections delve into text editing features of the UNIX operating system and describe programs that are used to format a document in a user controlled style.

The Document Preparation section contains three parts:

- UNIX System Facilities
- Advanced Editing
- Stream Editor

The Advanced Editing part describes text editing functions which include special characters, line addressing, global commands, commands for cut and paste operations, and text editor-based programs. The Stream Editor part describes the noninteractive context editor. The UNIX System Facilities part outines some other tools that aid in document preparation.

The Formatting Facilities section contains three parts:

- NROFF and TROFF User's Manual
- Table Formatting Program
- Mathematics Typesetting Program.

The NROFF and TROFF User's Manual part presents information to enable the user to do simple formatting tasks and to make incremental changes to existing packages of **troff** formatter commands. The UNIX operating system formatter, **nroff**, is identical to the **troff** formatter in most respects. The Table Formatting Program part describes the **tbl** program usage and the input commands used to generate documents that contain tables. The Mathematics Typesetting Program part describes the **eqn** program usage and language for obtaining text with mathematical expressions. The language interfaces directly

* Trademark of AT&T.

with the **troff** processor so mathematical expressions can be embedded in the running text of a manuscript and the entire document produced in one process. The Memorandum Macros section is a user's guide and reference manual for the Memorandum Macros (MM) package. These macros provide a general purpose package of text formatting macros used with the **nroff** and **troff** formatters. The macros provide users of the UNIX operating system a unified, consistent, and flexible tool for producing many common types of documents. Although the UNIX operating system provides other macro packages for various specialized formats, MM is the standard general purpose macro package for most documents such as letters, reports, technical memoranda, released papers, manuals, books, design proposals, and user guides. Uses of MM range from single-page letters to documents of several hundred pages in length.

The Viewgraphs and Slides Macros section describes the MV package of macros. These macros provide users a method of preparing viewgraphs and slides using the **troff** formatter. Included is a discussion on the use of the macros and a philosophy of how a viewgraph or slide should appear.

TABLE OF CONTENTS

VIEWGRAPHS AND SLIDES MACROS

LIST OF FIGURES AND TABLES

DOCUMENT PREPARATION

The following section deals with the basics of document preparation. The sections covered are:

- UNIX SYSTEM FACILITIES
- ADVANCED EDITING
- STREAM EDITOR

UNIX SYSTEM FACILITIES

Several miscellaneous facilities exist (via UNIX operating system commands) to aid in the development of documentation. These facilities are easy to access and are very effective. Their use is beneficial in documentation development. Some available miscellaneous facilities are described briefly in the following list. The *UNIX Programmer's Manual—Volume 1: Commands and Utilities* has a more detailed list.

bdiff The **bdiff** facility is used in a manner analogous to **diff** to find which lines must be changed in two files to bring them into agreement. Its purpose is to allow processing of files which are too large for **diff**.

cat The **cat** facility reads each file in sequence and writes it on the standard output. Thus:

cat *file*

prints the file named *file*, and

cat *file1 file2* >*file3*

concatenates *file1* and *file2* and places the result in *file3*.

cmp	The **cmp** facility compares two files. Under default options, **cmp** makes no comment if the files are the same; if they differ, it announces the byte and line number at which the difference occurred.
comm	The **comm** facility selects or rejects lines common to two sorted files. It reads *file1* and *file2* and produces a 3-column output as follows: lines only in *file1*, lines only in *file2*, and lines in both files.
diff	The **diff** facility is a differential file comparator. It tells what lines must be changed in two files to bring them into agreement.
diff3	The **diff3** facility is a 3-way differential file (files up to 64K) comparator. It compares three versions of a file and publishes disagreeing ranges of text flagged with special codes.
diffmk	The **diffmk** facility marks the differences between files. It compares two versions of a file and creates a third file that includes "change mark" commands for the **nroff** or **troff** formatter.
grep	Commands of the **grep** facility search the input files for lines matching a pattern. Normally, each line found is copied to the standard output. The **grep** patterns are limited regular expressions in the style of **ed**. The **egrep** patterns are full regular expressions. The **fgrep** patterns are fixed strings.
pr	The **pr** facility prints the named files on the standard output. If file is – or if no files are specified, the standard input is assumed.
sdiff	The **sdiff** facility uses the output of **diff** to produce a side-by-side listing of two files indicating those lines that are different. Each line of the two files are printed with a blank gutter between them if the lines are identical, a > in the gutter if the line exists only in *file1*, a < in the gutter if the line exists only in *file2*, and a | for lines that are different.
sort	The **sort** facility sorts lines of all the named files together and writes the results on the standard output.

spell	The **spell** facility collects words from the named files and looks them up in a spelling list. Words that do not occur in the spelling list nor can be derived from them are printed on the standard output. The **spellin** and **spellout** programs are two additional subroutines of **spell.**
split	The **split** facility splits a file into pieces.
typo	The **typo** facility searches through a document for unusual words, typographical errors, and hapax legomena and prints them on the standard output.
uniq	The **uniq** facility reports repeated lines in a file. It reads the input file comparing adjacent lines. In the normal case, the second and succeeding copies of repeated lines are removed; the remainder is written on the output file.

ADVANCED EDITING

Introduction

The advanced editing part is meant to help UNIX operating system users (secretaries, typists, programmers, etc.) make effective use of facilities for preparing and editing documents, text, programs, files, etc. It provides explanations and examples of:

- special characters, operating on lines, line addressing, and global commands in the text editor (**ed**)

- commands for "cut and paste" operations on files and parts of files, including **mv**, **cp**, **cat**, and **rm** commands, and **r**, **w**, **m**, and **t** commands of the text editor (**ed**)

- editing scripts and text editor-based programs like **grep** and **sed** {7}.

Examples are based on experience and observations of users and the difficulties encountered.

Although this document is written for non-programmers, new UNIX operating system users with any background should find helpful hints on how to get their jobs done more easily. The UNIX operating system provides tools for text editing, but that by itself is no guarantee that everyone will make the most effective use of them. In particular, users who are not computer specialists (typists, secretaries, casual users) often use the UNIX operating system less effectively than they could.

The next paragraphs discuss shortcuts and labor-saving devices. Not all will be instantly useful (some will) and others should provide ideas for future use.

A document like this should provide ideas about what to try. There is only one way to learn to use something, and that is to use it. Reading a description is no substitute for hands-on use.

The reader should be familiar with the **HOW TO GET STARTED** section of the *UNIX Programmer's Manual—Volume 1: Commands and Utilities* before using the text editor. Further information on all commands discussed here can be found in:

- *UNIX Programmer's Manual—Volume 1: Commands and Utilities*
- *UNIX Programmer's Manual—Volume 2: System Calls and Library Routines*
- *UNIX Programmer's Manual—Volume 3: System Administration Facilities*

Special Characters

The **ed** program is the primary interface to the system, so it is worthwhile to know how to get the most out of it with the least effort.

Print and List Commands

Two commands are provided for printing contents of lines being edited. Most users are familiar with the print command (**p**) in combinations like

```
1,$p
```

to print all lines that are being edited, or

```
s/abc/def/p
```

to change the first "abc" to "def" on the current line and print the results. Less familiar is the list command (**l**) which gives slightly more information than **p**. In particular, **l** makes characters visible that are normally invisible, such as tabs and backspaces. If a line contains some of these, the **l** command will print each tab as ">" and each backspace as "<". This makes it easier to correct typing mistakes that insert extra spaces adjacent to tabs or a backspace followed by a space.

The **l** command also "folds" long lines for display purposes. Any line that exceeds 72 characters is printed on multiple lines. Each printed line except the last is automatically terminated by a backslash (\) to indicate that the line was folded. A "$" character is appended to the real end of line. This is useful for printing long lines on terminals having output line capability of only 72 characters per line.

Occasionally, the **l** command will print a string of numbers preceded by a backslash, such as "\07" or "\16". These combinations are used to make characters visible that normally do not print, e.g., form feed, vertical tab, or bell. Each such combination is interpreted as a single character. When such characters are detected, they may have surprising meanings when printed on some terminals. Often their presence means that a finger slipped while typing.

Substitute Commands

The substitute command (**s**) is used for changing the contents of individual lines. It is probably the most complex and effective of any **ed** command.

The meaning of a trailing global command (**g**) after a substitute command is illustrated in the next two commands:

```
s/this/that/
```

and

```
s/this/that/g
```

The first form replaces the first "this" on the line with "that". If there is more than one occurrence of "this" on the line, the second form (with the trailing **g**) changes all of them. Either of the two forms of the **s** command can be followed by **p** or **l** to print or list the contents of the line:

```
s/this/that/p
s/this/that/l
s/this/that/gp
s/this/that/gl
```

All are legal and have slightly different meanings.

An **s** command can be preceded by one or two line numbers to specify that the substitution is to take place on a group of lines specified by the line numbers. Thus:

```
1,$s/mispell/misspell/
```

changes the first occurrence of "mispell" to "misspell" on every line of the file. The following command changes every occurrence on every line:

```
1,$s/mispell/misspell/g
```

By adding a **p** or **l** to the end of any of these substitute commands, only the last line that was changed will be printed, not all lines. How to print all the lines that were changed is described in a later section.

Any character can be used to delimit pieces of an **s** command. There is nothing sacred about slashes (but slashes must be used for context searching). For instance, for a line that contains a lot of slashes already, e.g.:

```
//exec //sys.fort.go // etc...
```

a colon could be used as the delimiter. To delete all the slashes, the command is

```
s/:/::g
```

Undo Command

Occasionally, an erroneous substitution is made in a line. The undo command (**u**) negates the last command so that data is restored to its previous state. This command is especially useful after executing a global command if it is discovered the command did things that are undesirable.

Metacharacters

When using **ed**, certain characters have special meanings when they occur in the left side of a substitute command or in a search for a particular line. These are called "metacharacters" which are:

- Period .
- Backslash \

- Dollar Sign **$**
- Circumflex ^
- Star *
- Brackets [ ]
- Ampersand **&**

Even though metacharacters are discussed separately in the following text, they can be combined.

Period

The period (**.**) on the left side of a substitute command or in a search stands for any single character. Thus the search

```
/x.y/
```

finds any line where "x" and "y" occur separated by a single character, as in

```
x+y
x–y
x<sp>y
x.y
```

The "**<sp>**" stands for a space character whenever needed to make it visible.

Since the period matches any single character, a way to deal with the "invisible" characters printed by **l** is available. For instance, if there is a line that when printed with the **l** command, appears as

```
...th\07is...
```

and it is desired to get rid of the "\07" (the bell character), the most obvious solution is to try

```
s/\07//
```

This will fail. Retyping the entire line is a reasonable tactic if the line in question is not too long. However, for a very long line, retyping could result in additional errors. Since "\07" really represents a single character, the command

```
s/th.is/this/
```

gets the job done. The period matches the mysterious character between the "h" and the "i".

Since the period matches any single character, the command

```
s/./,/
```

converts the first character on the line into a ",".

As is true of many characters in **ed**, the period has several meanings depending on its context. In the line

```
.s/././
```

• The first period is the line number of the line being edited, which is called "dot".

• The second period is a metacharacter that matches any single character on that line (in this instance the first character of the line).

• The third period is the only one that really is a literal period. On the right side of a substitution, the period is not special.

Backslash

Since a period means "any character", the question arises of what to do when a period is really needed. For example, to convert the line:

```
Now is the time.
```

into

```
    Now is the time?
```

the backslash (\) is used. A backslash turns off any special meaning that the next character might have. In particular, "\." converts the period from a "match anything" into a "match the period" statement. The "\." pair of characters is considered by **ed** to be a single literal period. To replace the period with a question mark, the following command is used:

```
    s/\./?/
```

The backslash can also be used when searching for lines that contain a special character. If a search is made to look for a line that contains

```
    .PP
```

the search

```
    /.PP/
```

is not adequate. It will find a line like

```
    THE APPLICATION OF ...
```

The period matches the letter "A". But if the command

```
    /\.PP/
```

is used, only the lines that contain ".PP" are found.

The backslash can also be used to turn off special meanings for characters other than the period. For example, to find a line that contains a backslash, the search

```
/\/
```

will not work because the "\" is not a literal backslash, but instead means that the second "/" no longer delimits the search. A search can be made for a literal backslash by preceding a backslash with another "\":

```
/\\/
```

Similarly, searches can be made for a slash (/) with

```
/\//
```

The backslash turns off the meaning of the immediately following "/" so that it does not terminate the search prematurely.

Some substitute commands, each of which will convert the line

```
\x\.\y
```

into the line

```
\x\y
```

are

```
s/\\\.//
s/x../x/
s/..y/y/
```

The user's erase character and the line kill character (# and @ by default) must also be used with a backslash to turn off their special meaning. When adding text with append (**a**), insert (**i**), or change (**c**) commands, the backslash is special only for the erase and line kill characters, and only one backslash should be used for each one needed.

Dollar Sign

In the left side of a substitute command or in a search command the dollar sign ($) stands for "the end of line". The word "time" is added to the end of the following phrase:

```
        Now is the
```

with the following command:

```
        s/$/<sp>time/
```

The result is

```
        Now is the time
```

A space is needed before "time" in the substitute command, otherwise, the following will be printed:

```
        Now is thetime
```

The second comma in the following line can be replaced with a period without altering the first:

```
        Now is the time, for all good men,
```

The needed command is

```
        s/,$/./
```

The "$" provides context to indicate which specific comma. Without it the s command would operate on the first comma to produce

```
        Now is the time. for all good men,
```

To convert

```
Now is the time.
```

into

```
Now is the time?
```

that was previously done with the backslash, the following command is used:

```
s/.$/?/
```

The "$" has multiple meanings depending on context. In the line

```
$s/$/$/
```

- The first **$** refers to the last line of the file.
- The second **$** refers to the end of the line.
- The third **$** is a literal dollar sign to be added to that line.

Circumflex

The circumflex (^), alias "hat" or "caret", stands for the beginning of the line. For example, if a search is made for a line that begins with "the", the command

```
/the/
```

will probably find several lines that contain "the" before arriving at the line that was wanted. But the command

```
/^the/
```

narrows the context, and thus arrives at the desired line more quickly.

The other use of the circumflex is to enable text to be inserted at the beginning of a line. For example:

```
s/^/<sp>/
```

places a space at the beginning of the current line.

Metacharacters can be combined. For example, to search for a line that contains only the characters

```
.PP
```

the command

```
/^\.PP$/
```

can be used.

Star

The star (*) is useful to replace all spaces between *x* and *y* with a single space, as in the following example:

```
        text x      y text
```

where *text* stands for lots of text, and there are an indeterminate number of spaces between *x* and *y*. The line is too long to retype, and there are too many spaces to count.

A regular expression (typically a single character) followed by a star stands for as many consecutive occurrences of that regular expression as possible. To refer to all the spaces at once, the following command is used:

```
s/x<sp>*y/x<sp>y/
```

The construction **<sp>*** means "as many spaces as possible". Thus **x<sp>*y** means: "an x, followed by as many spaces as possible, and then a y".

The star can be used with any character, not just space. If the original example was *text* x−−−−−−−−−y *text*

then all "−" characters can be replaced by a single space with the command

```
s/x−*y/x<sp>y/
```

If the original line was

text x.........y *text*

and if the following command was typed:

```
s/x.*y/x<sp>y/
```

what happens depends upon the occurrence of other x's or y's on the line. If there are no other x's or y's, then everything works, but it is blind luck, not good management. Since a period matches any single character, then **.*** matches as many single characters as possible. Unless the user is careful the star can eat up a lot more of the line than expected. If the line was

text x *text* x.........y *text* y *text*

then the command will take everything from the first "x" to the last "y", which, in this example, is undoubtedly more than wanted. The proper way is to turn off the special meaning of period with "\.".

```
s/x\.*y/x<sp>y/
```

Now everything works since "\.*" means "as many periods as possible".

There are times when the pattern ".*" is exactly what is wanted. For example, to change

Now is the time for all good men ...

into

Now is the time.

the following deletes everything after the word "time":

```
s/<sp>for.*/./
```

There are a couple of additional pitfalls associated with * to be aware of. Most notable is that "as many as possible" means zero or more. The fact that zero is a legitimate possibility is sometimes rather surprising. For example, if a line contains

text xy *text* x   y *text*

and the command is

```
s/x<sp>*y/x<sp>y/
```

the first "xy" matches this pattern, for it consists of an "x", zero spaces, and a "y". The result is that the substitute acts on the first "xy" and does not touch the later one that actually contains some intervening spaces. The proper way is to specify a pattern like

```
/x<sp><sp>*y/
```

which says "an x, a space, as many more spaces as possible, and then a y" (in other words, one or more spaces).

The other startling behavior of * is also related to the zero being a legitimate number of occurrences of something followed by a star. The command

```
s/x*/y/g
```

when applied to the line

```
abcdef
```

produces

```
yaybycydyeyfy
```

which is almost certainly not what was intended. The reason for this behavior is that zero is a legal number of matches, and there is no "x" at the beginning of the line (so that gets converted into a "y"), nor between the "a" and the "b" (so that gets converted into a "y"), etc. The following command:

```
s/xx*/y/g
```

where "xx*" is "one or more x's", when applied to the line

```
abcdefxghi
```

produces

```
abcdefyghi
```

Brackets

Should a number that appears at the beginning of all lines of a file need to be deleted, a first thought might be to perform a series of commands like:

```
1,$s/^1*//
1,$s/^2*//
1,$s/^3*//
    .
    .
```

This is going to take forever if the numbers are long. Unless it is desired to repeat the commands over and over until finally all numbers are gone, the digits can be deleted on one pass. This is the purpose of brackets ([ ]).

The construction

```
[0123456789]
```

matches any single digit. The whole thing is called a "character class". With a character class, the job is easy. The pattern "[0123456789]*" matches zero or more digits (an entire number), so

```
1,$s/^[0123456789]* //
```

deletes all digits from the beginning of all lines.

Any characters can appear within a character class; and just to confuse the issue, there are essentially no special characters inside the brackets. Even the backslash does not have a special meaning. The following command searches for special characters within the brackets:

```
/[.\$^[]/
```

Within a character class, the "[" is not special. To get a "]" into a character class, it should be placed as the first character in the class. For example:

```
/[].\$^[]/
```

It is a nuisance to have to spell out the digits. They can be abbreviated as [0–9]; similarly, [a–z] stands for the lowercase letters and [A–Z] for uppercase letters.

The user can specify a character class that means "none of the following characters". This is done by beginning the class with a circumflex.

```
[^0-9]
```

which stands for "any character except a digit". The following search finds the first line that does not begin with a tab or space:

```
/^[^(space)(tab)]/
```

Within a character class, the circumflex has a special meaning only if it occurs at the beginning. For example:

```
/^[^^]/
```

finds a line that does not begin with a circumflex.

Ampersand

The ampersand (**&**) is used primarily to save typing. For example, if the following is the original line:

```
Now is the time
```

and it needs to be

```
Now is the best time
```

the command

```
s/the/the best/
```

can be used, but it is unnecessary to repeat the "the". The **&** is used to eliminate the repetition. On the right-hand side of a substitute command, the ampersand means "whatever was just matched", so in the command

```
s/the/& best/
```

the **&** represents "the". This is not much of a saving if the thing matched is just "the"; but if it is something long or complicated or if it is something (such as **.***) which matches a lot of text, the **&** can save some tedious typing. There is also much less chance of making a typing error in the replacement text. For example, to parenthesize a line, regardless of its length:

```
s/.*/(&)/
```

The ampersand can occur more than once on the right side.

```
s/the/& best and & worst/
```

makes the original line

```
Now is the best and the worst time
```

and

```
s/.*/&? &!!/
```

converts the original line into

```
Now is the time? Now is the time!!
```

To get a literal ampersand, the backslash is used to turn off the special meaning.

```
s/ampersand/\& /
```

converts the word into the symbol. The **&** is not special on the left-hand side of a substitute, only on the right.

Operating On Lines

Substituting Newline Characters

The **ed** program provides a facility for splitting a single line into two or more lines by substituting in a newline character. If a line is unmanageably long because of editing or merely because of the way it is typed, it can be divided as follows:

text xy *text*

can be broken between the "x" and the "y" with the following substitute command:

```
s/xy/x\
y/
```

This is actually a single command although it is typed on two lines. Since the "\" turns off special meanings, a "\" at the end of a line makes the newline character there no longer special. When a new line is substituted in, dot is left pointing at the last line created.

A single line can be made into several lines with this same mechanism. The word "very" in the following example can be put on a separate line preceded with the **nroff** formatter underline command (**.ul**):

> *text* a very big *text*

The commands

```
s/<sp>very<sp>/\
.ul\
very\
/
```

convert the line into four shorter lines:

> *text* a
> .ul
> very
> big *text*

The word "very" is preceded by the line containing the ".ul" and spaces around "very" are eliminated at the same time.

Joining Lines

Lines may be connected with the join command (**j**). Given the lines

```
Now is
<sp>the time
```

and if dot is set to the first line, then the **j** command joins them together. No spaces are added which is why a space is shown at the beginning of the second line.

All by itself, a **j** command joins dot to dot+1. Any contiguous set of lines can be joined by specifying the starting and ending line numbers. For example:

```
1,$jp
```

joins all the lines into a big one and prints it.

Rearranging Lines

The **&** metacharacter stands for whatever was matched by the left side of an s command. Similarly, several pieces can be captured of what was matched; the only difference is it must be specified on the left side just what pieces the user is interested in. For instance, if there is a file of lines that consist of names in the form

```
Smith, A. B.
Jones, C.
```

etc., and it was intended to have the initials to precede the name, as in:

```
A. B. Smith
C. Jones
```

It is possible to do this with a series of tedious and error-prone editing commands. The alternative is to "tag" the pieces of the pattern (in this case, the last name and the initials) and then rearrange the pieces. On the left side of a substitution if part of the pattern is enclosed between "\(" and "\)", whatever matched that part is remembered and available for use on the right side. On the right side, the symbol "\1" refers to whatever matched the first "\(...\)" pair, "\2" to the second "\(...\)" pair, etc.

The command

```
1,$s/^\([^,]*\),<sp>*\(.*\)/\2<sp>\1/
```

although hard to read, does the job. The first "\(...\)" matches the last name, which is any string up to the comma; this is referred to on the right side with "\1". The second "\(...\)" is whatever follows the comma and any spaces and is referred to as "\2".

With any complicated editing sequence, it is foolhardy to run it and hope. Global commands provide a way to print those lines affected by the substitute command.

Line Addressing in the Editor

Line addressing in **ed** specifies the lines to be affected by editing commands. Previous constructions like

```
1,$s/x/y/
```

were used to specify a change on all lines. Most users are familiar with using a single newline character (or return) to print the next line and with

```
/string/
```

to find a line that contains "string". Less familiar is the use of

```
?string?
```

to scan backwards for the previous occurrence of "string". This is handy when the user realizes that the string to be operated on is back up the page (file) from the current line being edited. The slash and question mark are the only characters that can be used to delimit a context search.

Address Arithmetic

It is possible to combine line numbers like ".", "$", "/.../", and "?...?" with "+" and "−". Thus:

```
$−1
```

is a command to print the next to last line of the current file (i.e., one line before line $). For example:

```
$-5,$p
```

prints the last six lines. If there are not six lines, an error message will be indicated.

As another example:

```
.-3,.+3p
```

prints from three lines before the current line to three lines after. The "+" can be omitted:

```
.-3,.3p
```

is identical in meaning.

Another area in which to save typing effort in specifying lines is by using "−" and "+" as line numbers by themselves. For instance, a

```
-
```

by itself is a command to move back up one line in the file. Several minus signs can be strung together to move back up that many lines:

```
---
```

moves up three lines, as does "−3". Thus:

```
-3,+3p
```

is also identical to the examples above.

Since "−" is shorter than ".−1", constructions like

```
    −,.s/bad/good/
```

are useful. This changes the first occurrence of "bad" to "good" on both the previous line and the current line.

The "+" and "−" can be used in combination with searches using "/string/", "?string?", and "$". The search

```
    /string/−−
```

finds the line containing "string" and positions dot two lines before it.

Repeated Searches

When the search command is

```
    /horrible string/
```

and when the line is printed, it is discovered that it is not the "horrible string" that was wanted. It is necessary to repeat the search again, but it is not necessary to retype it. The construction

```
    //
```

is a shorthand for "the string that was previously searched for", whatever it was. This can be repeated as many times as necessary. This also applies to the backwards search

```
    ??
```

which searches for the same string but in the reverse direction.

Not only can the search be repeated, but the // construction can be used on the left side of a substitute command to mean "the most recent pattern":

```
/horrible string/
   --- ed prints line with "horrible string"
s//good/p
   --- ed prints the corrected line
```

To search backwards and change a line, the following command is used:

```
??s//good/
```

Of course, the **&** on the right-hand side of a substitute can still be used to stand for whatever got matched:

```
//s//&<sp>&/p
```

finds the next occurrence of whatever was searched for last, replaces it by two copies of itself, and then prints the line to verify that it worked.

Default Line Numbers

One of the most effective ways to speed editing is by knowing which lines are affected by a command with no address and where dot will be positioned when a command finishes. Editing without specifying unnecessary line numbers can save a lot of typing. As the most obvious example, the search command

```
/string/
```

puts dot at the next line that contains "string". No address is required with commands like:

- **p** to print the current line
- **l** to list the current line
- **d** to delete the current line
- **a** to append text after the current line
- **c** to change the current line

- **i** to insert text before the current line.

- **s** to make a substitution on the current line

If there was no "string" detected, dot stays on the line where it was. This is also true if it was sitting on the only "string" when the command was issued. The same rules hold for searches that use " ?string? "; the only difference is direction of search.

The delete command (**d**) leaves dot at the line following the last deleted line. However, dot points to the new last line when the last line is deleted.

Line-changing commands **a**, **c**, and **i** affect (by default) the current line if no line number is specified. They behave identically in one respect - after appending, changing, or inserting, dot points at the last line entered. For example, the following can be done without specifying any line number for the substitute command or for the second append command:

```
a
  --- text
  --- botch            (minor error)
.
s/botch/correct/       (fix botched line)
a
  --- more text
```

The following overwrites the major error and permits continuation of entering information:

```
a
  --- text
  --- horrible botch   (major error)
.
c
  --- fixed up line    (replace entire line)
  --- more text
```

The read command (**r**) will read a file into the text being edited, either at the end if no address is given or after the specified line if an address is given. In either case, dot points at the last line read in. The **0r** command can be used to read in a file at the beginning of the text and the **0a** or **1i** commands can be used to start adding text at the beginning.

The write command (**w**) writes out the entire file. If the command is preceded by one line number, that line is written. Preceding the command by two line numbers causes a range of lines to be written. The **w** command does not change dot, therefore, the current line remains the same regardless of what lines are written. This is true even if a command like

```
/^\.AB/,/^\.AE/w abstract
```

is made which involves a context search. Since the **w** command is easy to use, the text being edited should be saved regularly just in case the system crashes or a file being edited is clobbered.

The command with the least intuitive behavior is the **s** command. The dot remains at the last line that was changed. If there were no changes, then dot is unchanged. To illustrate, if there are three lines in the buffer and dot is sitting on the middle one:

```
x1
x2
x3
```

the command

```
−,+s/x/y/p
```

prints the third line, which is the last one changed. But if the three lines had been:

```
x1
y2
y3
```

and the same command issued while dot pointed at the second line, then the result would be to change and print only the first line and that is where dot would be set.

Semicolon

Searches with "/.../" and "?...?" start at the current line and move forward or backward, respectively, until they either find the pattern or get back to the current line. Sometimes this is not what is wanted. Suppose, for example, that the buffer contains lines like

```
.
.
.
ab
.
.
.
bc
.
.
.
```

Starting at line 1, one would expect that the command

```
/a/,/b/p
```

prints all the lines from the "ab" to the "bc", inclusive. This is not what happens. Both searches (for "a" and for "b") start from the same point, and thus they both find the line that contains "ab". The result is to print a single line. If there had been a line with a "b" in it before the "ab" line, then the print command would be in error since the second line number would be less than the first; and it is illegal to try to print lines in reverse order. This is because the comma separator for line numbers does not set dot while each address is processed. Each search starts from the same place.

In **ed**, the semicolon (;) can be used just like comma with the single difference that use of a semicolon forces dot to be set at that point while line numbers are being evaluated. In effect, the semicolon "moves" dot. Thus in the example above, the command

```
/a/;/b/p
```

prints the range of lines from "ab" to "bc" because after the "a" is found dot is set to that line, and then "b" is searched for starting beyond that line. This property is most often useful in a very simple situation. If the need is to find the second occurrence of "string", then the commands

```
/string/
//
```

print the first occurrence as well as the second. The command

```
/string/;//
```

finds the first occurrence of "string" and sets dot there. Then it finds the second occurrence and prints only that line.

Searching for the second previous occurrence of "string", as in

```
?string?;??
```

is similar. Printing the third, fourth, etc. occurrence in either direction is left as an exercise.

When searching for the first occurrence of a character string in a file where dot is positioned at an arbitrary place within the file, the command

```
1;/string/
```

will fail if "string" occurs on line 1. It is possible to use the command

```
0;/string/
```

(one of the few places where 0 is a legal line number) to start the search at line 1.

Interrupting the Editor

If the user interrupts **ed** while performing a command by depressing the BREAK key, the INTERRUPT key, or the user interrupt character (RUB OUT or DEL CHAR keys by default), the file is put back together again. The file state is restored as much as possible to what it was before the command began. Naturally, some changes are irrevocable. If the file is being read from or written into, substitutions are being made, or lines are being deleted, these will be stopped in some clean but unpredictable state in the middle of the command execution (which is why it is not usually wise to stop them). Dot may or may not be changed.

Printing is more clear cut. Dot is not changed until the printing is done. Thus if a user interrupts **ed** while some printing is being done, dot is not sitting on the last printed line or even near it. Dot is returned to where it was when the **p** command was started.

Global Commands

Basic

Global commands (**g** and **v**) are used to perform one or more editing commands on all lines of a file. The **g** command operates on those lines that contain a specified string. As the simplest example, the command

```
g/THIS/p
```

prints all lines that contain the string "THIS". The string that goes between the slashes can be anything that could be used in a line search or in a substitute command; exactly the same rules and limitations apply. As another example:

```
g/^\./p
```

prints all lines that begin with period.

The **v** command (there is no mnemonic significance to the letter "v") is identical to **g**, except that it operates on those lines that do not contain an occurrence of the string. So

```
v/^\./p
```

prints the lines that do not begin with period.

The command that follows **g** or **v** can be almost any command. For example:

```
g/^\./d
```

deletes all lines that begin with period, and

```
g/^$/d
```

deletes all empty (blank) lines.

Probably the most useful command that can follow a global command is the substitute command since this can be used to make a change and print each affected line for verification. For example, to change the word "This" to "THIS" everywhere in a file and verify that it really worked, the command is

```
g/This/s//THIS/gp
```

The use of "//" in the substitute command means "the previous pattern", in this case, "This". The **p** command is done on every line that matches the pattern, not just those on which a substitution took place.

Global commands operate by making two passes over the file. On the first pass, all lines that match the pattern are marked. On the second pass, each marked line in turn is examined, dot is set to that line, and the command executed. This means that it is possible for the command that follows a **g** or **v** to use addresses, set dot, etc., quite freely. For example:

```
g/^\.PP/+
```

prints the line that follows each ".PP" macro (the signal for a new paragraph in some formatting packages). The "+" means "one line past dot", and

```
g/topic/?^\.SH?1
```

searches for each line that contains "topic", scans backwards until it finds a line that begins ".SH" (a section heading) and prints the line that follows, thus showing the section headings under which "topic" is mentioned. Finally:

```
g/^\.EQ/+,/^\.EN/-p
```

prints all the lines that lie between lines beginning with the ".EQ" and ".EN" formatting macros.

The **g** and **v** commands can also be preceded by line numbers, in which case the lines searched are only those in the range specified.

Multiline

It is possible to do more than one command under the control of a global command although the syntax for expressing the operation is not especially natural or easy. As an example, suppose the task is to change "x" to "y" and "a" to "b" on all lines that contain "string". Then:

```
g/string/s/x/y/\
s/a/b/
```

is sufficient. The backslash signals the **g** command that the set of commands continues on the next line. It terminates on the first line that does not end with "\". A substitute command can not be used to insert a newline character within a **g** command.

The command

```
g/x/s//y/\
s/a/b/
```

does not work as expected. The remembered pattern is the last pattern that was actually executed, so sometimes it will be "x" (as expected) and sometimes it will be "a" (not expected). The desired pattern should be spelled out:

```
g/x/s/x/y/\
s/a/b/
```

It is also possible to execute **a**, **c**, and **i** commands (append, change, and insert) under a global command. As with other multiline constructions, all that is needed is to add a "\" at the end of each line except the last. Thus to add a **.nf** and **.sp** command before each ".EQ" line, the following is typed:

```
g/^\.EQ/i\
.nf\
.sp
```

There is no need for a final line containing a period to terminate the **i** command unless there are further commands being done under the global. On the other hand, it does no harm to put it in.

It is good practice, after each global command, to check that the command did only what was desired. Surprises sometimes happen. When they do occur, the **u** command (undo) is useful to negate what was done by the last command.

Cut and Paste

There are two editing areas in which "cut and paste" operations can be performed.

- Command functions
- Text editor functions

Most operations are actually easy if the task is defined and precautions are taken when entering the commands.

Command Functions

The UNIX operating system command functions perform the following:

- Change name of files

- Copy files
- Combine files
- Remove files

Change Name of Files

If there is a file named *oldname* and if it needs to be renamed to *newname*, the move command (**mv**) will do the job. It moves the file from one name to another (the target file), for example:

```
mv oldname newname
```

Note: If there is already a file with the new name, its contents will be overwritten with information from the other (*oldname*) file. The one exception is that a file **cannot** be moved to itself, therefore, the following command is illegal.

```
mv oldname oldname
```

Copy Files

Sometimes a copy of a file is needed while retaining the original file. This might be because a file needs to be worked on and yet have a back up in case something happens to the file. In any case, the copy is made with the copy command (**cp**). To make a copy of a file named *good*, the following command will place a copy in a file named *savegood*:

```
cp good savegood
```

Two identical copies of the file *good* exist. If *savegood* previously contained something, it is overwritten.

To get the file *savegood* back to its original filename, *good*, the following commands are used:

```
mv savegood good
```

if *savegood* is not needed anymore or

```
cp savegood good
```

to retain a copy of *savegood.*

In summary, **mv** renames a file; **cp** makes a duplicate copy. Both commands overwrite the target file if one already exists unless write permission is denied by the mode of the file.

Combine Files

A familiar requirement is that of collecting two or more files into one big file, *bigfile*. This is needed, for example, when the author of a paper decides that several sections are to be combined. There are several ways to do this; the cleanest is a command called **cat** (not all commands have 2-letter names). The word **cat** is short for "concatenate" which is exactly what is desired. The command

```
cat file
```

prints the contents of the *file* on the terminal. The command

```
cat file1 file2
```

causes the contents of *file1* and *file2* to be printed on the terminal, in that order, but does not place them in *bigfile.*

There is a way to tell the system to put the same information in a file instead of printing on the terminal. The way to do it is to add to the command line the ">" character and the name of the file where the output is to go. The command

```
cat file1 file2 >bigfile
```

is used and the job is done. As with **cp** and **mv**, when something is put into *bigfile*, anything already there is destroyed. The ability to capture the output of a program can be used with any command that prints on a terminal.

Several files can be combined, not just two:

```
cat file1 file2 file3 ... >bigfile
```

collects many individual files.

Sometimes a file needs to be appended to the end of another file. For example:

```
cat good good1 >temp
mv temp good
```

is the most direct way. The following command:

```
cat good good1 >good
```

does not work because the ">" empties *good* before the **cat** program begins. The easiest way is to use a variant of ">", called ">>". In fact, ">>" is identical to ">" except that instead of clobbering the old file it adds something to the end. Thus the command

```
cat good1 >>good
```

adds *good1* to the end of *good*. If *good* does not exist, this makes a copy of *good1* called *good*.

Remove Files

If a file is not needed, it can be removed. The **rm** command

```
rm savegood
```

irrevocably deletes the file called *savegood* if the user has write permission.

Several files can be removed with one command.

```
rm save*
```

removes all files that begin with “save”. The command

rm *save1 save2 save3*

removes three files.

Text Editor Functions

The text editor (**ed**) function performs the following operations:

- Insert one file into another
- Write out part of a file
- Move lines around
- Copy lines
- Marks

File Names

It is important to know the editor (**ed**) commands for reading and writing files. Equally useful is the edit command (**e**). Within **ed**, the command

e *newfile*

says “edit a new file called *newfile* without leaving the text editor”. The **e** command discards whatever is being worked on and starts over on *newfile*. This is the same as if one had quit with the **q** command and reentered **ed** with a new file name except that if a pattern has been remembered, a command like “//” will still work.

When entering **ed** with the command

ed *file*

ed remembers the name of the file, and any subsequent **e**, **r**, or **w** commands that do not contain a file name will refer to this remembered file. Thus:

```
ed file1
   ---       (editing)
w            (writes back in file1)
e file2      (edit different file without leaving ed)
   ---       (editing on file2)
w            (writes back in file2)
```

etc. does a series of edits on various files without leaving **ed** and without typing the name of any file more than once. By examining the sequence of commands in this example, it can be seen why many operating systems use **e** as a synonym for **ed**.

The current file name can be found at any time with the **f** command by typing **f** without a file name. Also, the name of a remembered file can be changed with **f**. A useful sequence is

```
ed precious
f junk
   ---   (editing)
```

This obtains a copy of the file *precious* and guarantees that a subsequent **w** command without a filename will write to *junk* and will not overwrite the original file.

Insert One File Into Another

When a file is to be inserted into another, the **r** command can be used. For example, if the file *table* is to be inserted just after the reference to "Table 1", the following can be used:

```
/Table 1/
Table 1 shows that...       (response from ed)
.r table
```

The critical line is the last one. The **.r** command reads a file in after dot. An **r** command without any address adds lines to the end of the file, so it is equivalent to the **$r** command.

Write Out Part of a File

Another feature is writing to another file part of the document that is being edited. For example, it is possible to split into a separate file the table from the previous example, so it can be formatted or tested separately. If in the file being edited, there is

```
    --- text
.TS
    --- data for table
.TE
    --- text
```

(which is the way a table is set up as explained in the **TABLE FORMATTING PROGRAM** section) to isolate the table information in a separate file called *table*. First the start of the table (the **.TS** line) is found, and then the desired part is written on file *table*:

```
/^\.TS/
.TS                (response from ed)
.,/^\.TE/w table
```

The same job can be accomplished with the single command

```
/^\.TS/;/^\.TE/w table
```

The point is that the **w** command can write out a group of lines instead of the whole file. A single line can be written by using one line number instead of two. For example, if a complicated line was just typed and it will be needed again, it should be saved and read in later rather than retyped:

```
a
   --- lots of stuff
   --- stuff to repeat
.
.w temp
a
   --- more stuff
.
.r temp
a
   --- more stuff
.
```

Move Lines Around

Moving a paragraph from its present position in a paper to the end can be done several ways. For example, it is assumed that each paragraph in the paper begins with the ".PP" formatting macro. The brute force way (not necessarily bad) is to write the paragraph onto a temporary file, delete it from its current position, and then read in the temporary file at the end. If dot is at the ".PP" macro that begins the paragraph, this is the sequence of commands:

```
.,/^\.PP/−w temp
.,//−d
$r temp
```

This states that from where dot is now until one line before the next ".PP" write onto file *temp*. The same lines are deleted and the file *temp* is read in at the end of the working file.

An easier way is to use the move command (**m**) that **ed** provides. This does the whole set of operations at one time without a temporary file. The **m** command is like many other **ed** commands in that it takes up to two line numbers in front to tell which lines are to be affected. It is also followed by a line number that tells where the lines are to go. Thus:

```
line1,line2m line3
```

says "move all the lines from **line1** through **line2** to after **line3**". Any of "line1", etc., can be line numbers, strings between slashes or dollar signs, or other line specifications. If dot is at the first line of the paragraph, the command

```
.,/^\.PP/−m$
```

will also accomplish this task.

As another example of a frequent operation, the order of two adjacent lines can be reversed by moving the first one after the second. If dot is positioned at the first line, then

```
m+
```

does it. It says to move the line to after the dot. If dot is positioned on the second line:

```
m−−
```

does the interchange.

The **m** command is more concise and direct than writing, deleting, and rereading. The main difficulty with the **m** command is that if patterns are used to specify both the line being moved and the target line, they must be specified properly or the wrong lines may be moved. The result of a botched **m** command can be a costly mistake. Doing the job a step at a time makes it easier to verify that each step accomplished what was wanted. It is also a good idea to issue a **w** command before doing anything complicated; then if an error is made, it is easy to back up.

Copy Lines

The **ed** program provides a transfer command (**t**) for making a copy of a group of one or more lines at any point. This is often easier than writing and reading. The **t** command is identical to the **m** command except instead of moving lines it duplicates them at the place referenced. Thus:

```
1,$t$
```

duplicates the entire contents that is being edited. A more common use for **t** is creating a series of lines that differ only slightly. For example:

```
a
   --- long line of stuff
.
t.          (make a copy)
s/x/y/      (change it a bit)
t.          (make third copy)
s/y/z/      (change it a bit)
```

Marks

The **ed** program provides for marking a line with a particular name so that the line can be referenced later by its name regardless of its line number. This can be useful for moving lines and for keeping track of them as they move. The mark command is **k**. The mark name must be a single lowercase letter. The command

```
kx
```

marks the current line with the name "x". If a line number precedes the **k**, that line is marked. The marked line can then be referred to with the address

```
'x
```

Marks are most useful for moving things around. The first line of the block to be moved is found and marked with **ka**. Then the last line of the block is found and marked with **kb**. Dot is then positioned at the place where the lines are to go and the following command is performed:

```
'a,'bm
```

Note: Only one line can have a particular mark name associated with it at any given time.

Temporary Escape

Sometimes it is convenient to temporarily escape from the text editor to do some UNIX operating system command without leaving the text editor. The escape command (!) provides a way to do this. If the command

```
!<any UNIX operating system command>
```

is entered, the current editing state is suspended; and the command asked for is executed. When the command finishes, **ed** will return a signal by printing another "!" and editing can be resumed.

Any UNIX operating system command may be performed including another **ed** (this is quite common). In this case, another "!" can be done.

Supporting Tools

There are several related tools and techniques which are relatively easy to learn after **ed** has been learned because they are based on **ed**. Examples of these techniques are shown, more to indicate their existence than to provide a complete tutorial.

Global Printing From a Set of Files (grep)

Sometimes all occurrences of some word or pattern in a set of files need to be found in order to edit them or perhaps to verify their presence or absence. It may be possible to edit each file separately and look for the pattern of interest. If there are many files, this can be tedious; and if the files are really big, it may be impossible because of limits in **ed**.

The **grep** program was written to get around these limitations. Search patterns are often called "regular expressions", and "grep" stands for

```
g/re/p
```

This describes what **grep** does - it prints every line in a set of files that contains a particular pattern. Thus:

grep 'string' *file1 file2 file3* ...

finds "string" wherever it occurs in any of the files *file1*, *file2*, etc. The **grep** program also indicates the file in which the line was found, so it can be edited later if needed.

The pattern represented by "string" can be any pattern that can be used in the text editor since **grep** and **ed** use the same mechanism for pattern searching. It is wisest to enclose the pattern in the single quotes ('string') if it contains any nonalphabetic characters since many such characters also mean something special to the UNIX operating system command interpreter (the "shell"). Without single quotes, the command interpreter will try to interpret them before **grep** has the opportunity.

There is also a way to find lines that do not contain a pattern:

grep −v 'string' *file1 file2* ...

finds all lines that do not contain "string". The **−v** option must occur in the position shown. Given **grep** and **grep −v**, it is possible to select all lines that contain some combination of patterns. For example, to obtain all lines that contain "x" but not "y":

grep x *file...* | **grep** −v y

The pipe notation (|) causes the output of the first command to be used as input to the second command.

Editing Scripts

If a fairly complicated set of editing operations is to be performed on an entire set of files, the easiest thing to do is to make a script file, i.e., a file that contains the operations to be performed and then apply this script to each file in turn. For example, if every instance of "This" needs to be changed to "THIS" and every instance of "That" needs to be changed to "THAT" in a large number of files, a file *script* is made with the following contents:

```
g/This/s//THIS/g
g/That/s//THAT/g
w
q
```

The following is done:

```
ed file1 <script
ed file2 <script
...
```

This causes **ed** to take its commands from the prepared script. The whole job has to be planned in advance.

By using the UNIX operating system command interpreter [**sh**(1)], a set of files can be cycled automatically with varying degrees of ease.

STREAM EDITOR

Introduction

The stream editor (**sed**) is a noninteractive text editor that runs on the UNIX operating system. It is a lineal descendant of the text editor, **ed**. Changes between **ed** and **sed** exist because of differences between interactive and noninteractive operations. The **sed** software is especially useful in the following cases:

- Editing files that are too large for comfortable interactive editing

- Editing any size file when the sequence of editing commands is too complicated to be comfortably typed in interactive mode

- Performing multiple global editing functions efficiently in one pass through the input file.

The effective size of a file that can be edited is limited only by the requirement that input and output files fit simultaneously into available secondary storage. This is because only a few lines of the input file reside in memory at one time and no temporary files are used.

Complicated editing scripts can be created separately and given to the **sed** program as a command file. For complex edits, this saves considerable typing and attendant errors. The **sed** program running from a command file is more efficient than an interactive editor even if that editor can be driven by a prewritten script.

Principal loss of functions, compared to an interactive editor, are:

- Lack of relative addressing (because of the line-at-a-time operation)

- Lack of immediate verification that a command has done what was intended.

Overall Operation

The **sed** program by default copies the standard input to the standard output, perhaps performing one or more editing commands on each line before writing it to the output. This behavior may be modified by arguments on the command line.

Command Line

The general format of an editing command is

```
[address1,address2] function [arguments]
```

• One or both addresses may be omitted.

• The function must be present.

• Any number of blanks or tabs may separate addresses from the function.

• Arguments may be required or optional according to the function given. Three arguments are recognized on the command line:

−n	Copy only those lines specified by **p** (print) functions or **p** arguments after **s** (substitute) functions
−e	Take the next argument as an editing command
−f	Take the next argument as a file name; the file should contain editing commands — one to a line.

• Tab characters and spaces at the beginning of lines are ignored.

Order of Application of Editing Commands

All editing commands are compiled into a form which will be moderately efficient during the execution phase (when the commands are actually applied to lines of the input file) and before any input file is opened.

• Commands are compiled in the order encountered; generally, the order they will be attempted at execution time.

• Commands are applied one at a time; the input to each command is the output of all preceding commands.

The default linear order of application of editing commands can be changed by the **t** (test substitution) and **b** (branch) flow-of-control commands. When the order of application is changed by these commands, it remains true that the input line to any command is the output of any previously applied command.

Pattern Space

The range of pattern matches is called the pattern space. Ordinarily, pattern space is one line of the input text, but more than one line can be read into the pattern space by using the next command (**n**).

Examples

Examples scattered throughout the following paragraphs use the following standard input text, except where noted:

```
In Xanadu did Kubla Khan
A stately pleasure dome decree:
Where Alph, the sacred river, ran
Through caverns measureless to man
Down to a sunless sea.
```

The command

```
2q
```

will copy the first two lines of the input and quit. The output will be

```
In Xanadu did Kubla Khan
A stately pleasure dome decree:
```

Selecting Lines for Editing

Input file lines that editing commands are to be applied can be selected by addresses. Addresses may be either line numbers or context addresses.

The application of a group of commands can be controlled by one address (or address pair) by grouping commands with braces ({}).

Line Number Addresses

A line number is a decimal integer. As each line is read from the input, a line number counter is incremented. A line number address matches (selects) the input line causing the internal counter to equal the address line number. The counter runs cumulatively through multiple input files. It is not reset when a new input file is opened. As a special case, the **$** character matches the last line of the last input file.

Context Addresses

A context address is a pattern (a regular expression) enclosed in slashes (/string/). Regular expressions recognized by the **sed** program are constructed as follows:

• An ordinary character is a regular expression and matches that character.

• A circumflex (^) at the beginning of a regular expression matches the null character at the beginning of a line.

• A dollar sign ($) at the end of a regular expression matches the null character at the end of a line.

• The "\n" character matches an embedded newline character but not the newline character at the end of the pattern space.

• A period (.) matches any character except the terminal newline character of the pattern space.

• A regular expression followed by an asterisk (*) matches any number (including 0) of adjacent occurrences of the regular expression it follows.

• A string of characters in square brackets ([ ]) matches any character in the string and no others. If, however, the first character of the string is a circumflex (^), the regular expression matches any character except the characters in the string and the terminal newline character of the pattern space. The circumflex is the only metacharacter recognized within the square brackets. If "]" needs to be in the set of square brackets, it should be the first nonmetacharacter. For example:

[]...]	Includes "]"
[^]...]	Does not include "]"

• A concatenation of regular expressions is a regular expression which matches the concatenation of strings matched by the components of the regular expression.

• A regular expression between the sequences "\(" and "\)" is identical in effect to the unadorned regular expression but has side effects which are described under the s command (substitute function) below.

• The expression "\d" means the same string of characters matched by an expression enclosed in "\(" and "\)" earlier in the same pattern. The *d* is a single digit; the string specified is that beginning with occurrence *d* of "\(" counting from the left. For example, the following expression matches a line beginning with two repeated occurrences of the same string:

```
^\(.*\)\1
```

• The null regular expression standing alone (e.g., //) is equivalent to the last regular expression compiled.

• Special characters (^ $. * [ ] \ /), when used as literal characters, must be preceded by a backslash (\).

• For a context address to match, the whole pattern within the input address must match some portion of the pattern space.

Number of Addresses

Commands can have 0, 1, or 2 addresses. The maximum number of allowed addresses is given under each command. It is considered an error when a command has more addresses than the maximum allowed.

If a command has *no addresses*, it is applied to every line in the input.

If a command has *one address*, it is applied to all lines which match that address.

If a command has *two addresses*, it is applied to the first line which matches the first address and to all subsequent lines until (and including) the first subsequent line which matches the second address. An attempt is made on subsequent lines to again match the first address, and the process is repeated. Two addresses are separated by a comma. The following table indicates some command examples applied to the standard input text and the resulting match.

/an/	matches lines 1, 3, and 4
/an.*an/	matches line 1
/^an/	matches no lines
/./	matches all lines
/\./	matches line 5
/r*an/	matches lines 1, 3, and 4 (number = 0)
/\(an\).*\1/	matches line 1.

Functions

Functions are named by a single alphabetic character. In the following function summaries, the maximum number of allowable addresses is enclosed in parentheses followed by the single character function name. Possible arguments are enclosed in angle brackets (< >), and a description of each function is given. Angle brackets around arguments are not part of the argument and should not be typed in actual editing commands.

Whole Line Oriented Functions

(2)**d** The **d** function deletes from the file (does not write to the output) those lines matched by its addresses. It also has the side effect that no further commands are attempted on the corpse of a deleted line. As soon as the **d** function is executed, a new line is read from the input, and the list of editing commands is restarted from the beginning on the new line.

(2)**n** The **n** function reads the next line from the input, replacing the current line, and the current line is written to the output. The list of editing commands is continued following the **n** command.

(1)**a**\
<text> The **a** function causes the text argument (<text>) to be written to the output after the line matched by its address. The **a** command is inherently multiline; **a** must appear at the end of a line, and <text> may contain any number of lines. To preserve the one-command-to-a-line fiction, interior newline characters must be hidden by a backslash character (\) immediately preceding the newline character. The <text> is terminated by the first unhidden newline character not immediately preceded by a backslash. Once an **a** function is successfully executed, text will be written to the output regardless of what later commands do to the line which triggered it. Even if that line is deleted, text will still be written to the output. The <text> is not scanned for address matches, and no editing commands are attempted on it. The **a** function does not cause a change in the line number counter.

(1)**i**\
<text> The **i** function causes the text argument (<text>) to be written to the output before the line matched by its address. The **i** command is inherently multiline; **i** must appear at the end of a line, and <text> may contain any number of lines. To preserve the one-command-to-a-line fiction, interior newline characters must be hidden by a backslash character (\) immediately preceding the newline character. The <text> is terminated by the first unhidden newline character not immediately preceded by a backslash. Once an **i** function is successfully executed, text will be written to the output regardless of what later commands do to the line which triggered it. Even if that line is deleted, text will still be written to the output. The <text> is not scanned for address matches, and no editing commands are attempted on it. The **i** function does not cause a change in the line number counter.

(2)**c**\
<text> The **c** function deletes lines selected by its addresses and replaces them with the lines in the text argument (<text>). Like **a** and **i**, **c** must be followed by a newline character hidden by a backslash; interior newline characters in <text> must be hidden by backslashes. The **c** command may have two addresses, and therefore select a range of lines. If it does, all lines in the range are deleted, but only one copy of text is written to the output, not one copy per line deleted. As with **a** and **i**, <text> is not scanned for address matches, and no editing commands are attempted on it. It does not change the line number counter. After a line has been deleted by a **c** function, no further commands are attempted on the

corpse. If text is appended after a line by **a** or **r** functions and the line is subsequently changed, the text inserted by the **c** function will be placed before the text of the **a** or **r** functions (the **r** function is described later).

Leading blanks and tabs will disappear, as in **sed** commands, for text put in the output by these functions. To get leading blanks and tabs into the output, the first desired blank or tab is preceded by a backslash. The backslash will not appear in the output. The list of editing commands for example:

```
n
a\
XXXX
d
```

applied to the standard input produces

```
In Xanadu did Kubla Khan
XXXX
Where Alph, the sacred river, ran
XXXX
Down to a sunless sea.
```

In this particular case, the same effect would be produced by either of the two following command lists:

```
n
i\
XXXX
d
```

or

```
n
c\
XXXX
```

Substitute Function

One important substitute function that changes parts of lines selected by a context search within the line is

```
(2)s<pattern><replacement><flags>
```

The **s** function replaces the part of a line selected by <pattern> with <replacement>. It can be read

Substitute for <pattern>, <replacement>

Pattern

The pattern argument (<pattern>) contains a pattern exactly like the patterns in addresses. The only difference between <pattern> and a context address is that the context address must be delimited by slash (/) characters; <pattern> may be delimited by any character other than space or newline. By default, only the first string matched by <pattern> is replaced unless the **g** flag (below) is invoked.

Replacement

The replacement argument (<replacement>) begins immediately after the second delimiting character of <pattern> and must be followed immediately by another instance of the delimiting character (thus there are exactly three instances of the delimiting character). The <replacement> is not a pattern, and the characters which are special in patterns do not have special meaning in <replacement>. Instead, other characters are special:

\& is replaced by the string matched by <pattern>.

\d is replaced by substring **d** (**d** is a single digit), matched by parts of <pattern>, and enclosed in "\(" and "\)". If nested substrings occur in <pattern>, substring **d** is determined by counting opening delimiters (\(). As in patterns, special characters may be made literal characters by preceding them with backslash (\).

Flags

The flags argument (<flags>) may contain the following:

g Substitute <replacement> for all nonoverlapping instances of <pattern> in the line. After a successful substitution, the scan for the next instance of <pattern> begins just after the end of the inserted characters. Characters put into the line from <replacement> are not rescanned.

p Print the line if a successful replacement was done. The **p** flag causes the line to be written to the output if and only if a substitution was actually made by the **s** function. If several **s** functions, each followed by a **p** flag, successfully substitute in the same input line, multiple copies of the line will be written to the output — one for each successful substitution.

w <filename>
Write the line to a file if a successful replacement was done. The **w** flag causes lines which are actually substituted by the **s** function to be written to a file named by <filename>. If <filename> exists before **sed** is run, it is overwritten; if not, it is created. A single space must separate **w** and <filename>. The possibilities of multiple, somewhat different copies of one input line being written are the same as for **p**. A maximum of ten different file names may be mentioned after **w** flags and **w** functions.

Examples

The command

```
s/to/by/w changes
```

applied to the standard input produces on the output

```
In Xanadu did Kubla Khan
A stately pleasure dome decree:
Where Alph, the sacred river, ran
Through caverns measureless by man
Down by a sunless sea.
```

and on the file named *changes*

```
Through caverns measureless by man
Down by a sunless sea.
```

If the no-copy option is in effect, the command

```
s/[.,;?:]/*P&*/gp
```

produces

```
A stately pleasure dome decree*P:*
Where Alph*P,* the sacred river*P,* ran
Down to a sunless sea*P.*
```

To illustrate the effect of the **g** flag, the command

```
/X/s/an/AN/p
```

produces (assuming no-copy mode)

```
In XANadu did Kubla Khan
```

and the command

```
/X/s/an/AN/gp
```

produces

```
In XANadu did Kubla KhAN
```

Input/Output Functions

(2)**p** The *print* function writes addressed lines to the standard output file. They are written at the time the **p** function is encountered regardless of what succeeding editing commands may do to the lines.

(2)**w** <filename>
The *write* function writes addressed lines to the file named by

<filename>. If the file previously existed, it is overwritten; if not, it is created. The lines are written exactly as they exist when the write function is encountered for each line regardless of what subsequent editing commands may do to them. Exactly one space must separate the **w** and <filename>. A maximum of ten different files may be mentioned in write functions and **w** flags after **s** functions combined.

(1)**r** <filename>

The *read* function reads the contents of <filename> and appends them after the line matched by the address. The file is read and appended regardless of what subsequent editing commands do to the line which matched its address. If **r** and **a** functions are executed on the same line, the text from **a** functions and **r** functions is written to the output in the order that the functions are executed. Exactly one space must separate the **r** and <filename>. If a file mentioned by an **r** function cannot be opened, it is considered a null file, not an error, and no diagnostic is given.

Note: Since there is a limit to the number of files that can be opened simultaneously, care should be taken that no more than ten files be mentioned in **w** functions or flags. That number is reduced by one if any **r** functions are present (only one read file is opened at a time).

If the file *note1* has the following contents:

```
Note: Kubla Khan (more properly Kublai Khan;
1216-1294) was the grandson and most eminent
successor of Genghiz (Chingiz) Khan and founder
of the Mongol dynasty in China.
```

then the command

```
/Kubla/r note1
```

produces

```
In Xanadu did Kubla Khan
          Note: Kubla Khan (more properly Kublai Khan;
          1216-1294) was the grandson and most eminent
          successor of Genghiz (Chingiz) Khan and founder of
          the Mongol dynasty in China.
A stately pleasure dome decree:
Where Alph, the sacred river, ran
Through caverns measureless to man
Down to a sunless sea.
```

Multiple Input Line Functions

Three functions, all spelled with capital letters, deal with pattern spaces containing embedded newline characters. They are intended principally to provide pattern matches across lines in the input.

(2)**N** Append the next input line to the current line in the pattern space.
The two input lines are separated by an embedded newline character. Pattern matches may extend across embedded newline characters.

(2)**D** Delete first part of the pattern space.
Delete up to and including the first newline character in the current pattern space. If the pattern space becomes empty (the only newline character was the terminal newline character), read another line from the input. In any case, begin the list of editing commands again from the beginning.

(2)**P** Print first part of the pattern space.
Print up to and including the first newline character in the pattern space.

The **P** and **D** functions are equivalent to their lowercase counterparts if there are no embedded newline characters in the pattern space.

Hold and Get Functions

Four functions save and retrieve part of the input for possible later use.

(2)**h** Hold pattern space.
The **h** function copies contents of the pattern space into a hold area destroying previous contents.

(2)**H** Hold pattern space.
The **H** function appends contents of the pattern space to contents of the hold area. Former and new contents are separated by a newline character.

(2)**g** Get contents of hold area.
The **g** function copies contents of the hold area into the pattern space destroying previous contents.

(2)**G** Get contents of hold area.
The **G** function appends contents of the hold area to contents of the pattern space. Former and new contents are separated by a newline character.

(2)**x** Exchange.
The exchange command interchanges contents of the pattern space and the hold area.

The following are examples:

```
1h
1s/ did.*//
1x
G
s/\n/ :/
```

when applied to the standard input text produce

```
In Xanadu did Kubla Khan  :In Xanadu
A stately pleasure dome decree:  :In Xanadu
Where Alph, the sacred river, ran  :In Xanadu
Through caverns measureless to man  :In Xanadu
Down to a sunless sea.  :In Xanadu
```

Flow of Control Functions

These functions do no editing on the input lines but control the application of functions to the lines selected by the address part.

(2)**!** Don't.
The don't command causes the next command (written on the same

line) to be applied to those input lines not selected by the address part.

(2) Grouping.
The grouping command causes the next set of commands to be applied (or not applied) as a block to the input lines selected by the addresses of the grouping command. The first of the commands under control of the grouping may appear on the same line as the { or on the next line. The group of commands is terminated by a matching } standing on a line by itself. Groups can be nested.

(0):<label>
Place a label.
The label function marks a place in the list of editing commands which may be referred to by **b** and **t** functions. The <label> argument may be any sequence of eight or fewer characters. If two different colon functions have identical labels, a compile time diagnostic will be generated; and no execution attempted.

(2)**b**<label>
Branch to label.
The branch function causes the sequence of editing commands being applied to the current input line to be restarted immediately after the place where a colon function with the same <label> was encountered. If no colon function with the same label can be found after all editing commands have been compiled, a compile time diagnostic is produced; and no execution is attempted. A **b** function with no <label> is a branch to the end of the list of editing commands. Whatever should be done with the current input line is done, and another input line is read. The list of editing commands is restarted from the beginning on the new line.

(2)**t**<label>
Test substitutions.
The **t** function tests whether any successful substitutions have been made on the current input line; if so, it branches to <label>; if not, it does nothing. The flag which indicates that a successful substitution has been executed is reset by reading a new input line and executing a **t** function.

Miscellaneous Functions

(1) = The "=" function writes to standard output the line number of the line matched by its address.

(1)**q** The **q** function causes the current line to be written to the output (if it should be), any appended or read text to be written, and execution to be terminated.

NROFF AND TROFF USER'S MANUAL

INTRODUCTION

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

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

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

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

USAGE

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

> **nroff** *options files*
> or
> **troff** *options files*

where *options* represents any of a number of option arguments and *files* represents the list of files containing the document to be formatted. An argument consisting of a single minus sign (−) is taken to be a file name corresponding to the standard input. Input is taken from the standard input if no file names are given. Options may appear in any order so long as they appear before the files.

nroff AND troff OPTIONS

Option	*Effect*
−o*list*	Prints only pages whose page numbers appear in *list*, which 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 and cause the bell control character to be output to the terminal. The **nroff** formatter will halt after every *N*

pages (default $N=1$) to allow paper loading or changing and will resume upon receipt of a new line. The **troff** formatter will stop the phototypesetter every *N* pages, produce a trailer to allow changing cassettes, and resume after the phototypesetter START button is pressed.

−m*name* Prepend the macro file

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

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

−c*name* Prepend the macro files

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

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

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

nroff ONLY OPTIONS

Option	*Effect*
−T*name*	Specify the name of the output terminal type. Currently defined names are: • **37** (default) for the TELETYPE® Model 37 • **tn300** for the GE TermiNet 300 (or any terminal without half-line capabilities) • **300** for the DASI 300 • **300s** for the DASI 300s • **450** for the DASI 450.
−e	Produce equally spaced words in adjusted lines using full terminal resolution.
−h	Use output tabs during horizontal spacing to speed output and to reduce output byte count. Device tab settings are assumed to be every eight nominal character widths. The default settings of logical input tabs are also every eight nominal character widths.
−u*n*	Set the emboldening factor (number of character overstrikes) in the **nroff** formatter for the third font position (bold) to be *n* (zero if *n* is missing).

troff ONLY OPTIONS

Option	*Effect*
−t	Direct output to the standard output instead of the phototypesetter.
−f	Refrain from feeding paper and stopping phototypesetter at the end of the run.
−w	Wait until phototypesetter is available if busy.

−b	Report whether phototypesetter is busy or available. No text processing is done.
−a	Send a printable approximation in American Standard Code for Information Interchange (ASCII) character set of the results to the standard output. This approximates a display of the document.
−p*N*	Print all characters in point size *N* while retaining all prescribed spacings and motions to reduce phototypesetter elapsed time.
−g	Prepare output for the Murray Hill Computation Center phototypesetter and direct it to the standard output.

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

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

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

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

• The equation preprocessors are **neqn** and **eqn** (for **nroff** and **troff** formatters, respectively).

• The table-construction preprocessor is **tbl**.

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

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

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

```
tbl files | eqn | troff −t [options] | tc
```

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

NROFF/TROFF REFERENCE MANUAL

GENERAL EXPLANATION

Form of Input

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

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 a *x* for a single letter register name or (*xx* for a 2-character register name. Table B itemizes escape sequences for characters, indicators, and functions.

Formatter and Device Resolution

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

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

Numerical Parameter Input

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

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

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

• **em** for horizontally oriented requests (**.ll**, **.in**, **.ti**, **.ta**, **.lt**, **.po**, **.mc**) and functions (\h, \l).

• V for vertically oriented requests (**.pl**, **.wh**, **.ch**, **.dt**, **.sp**, **.sv**, **.ne**, **.rt**) and functions (\v, \x, \L)

- **p** for **.vs** request

- **u** for **.nr**, **.if**, and **.ie** requests.

All other requests ignore scale indicators. When a number register containing an already appropriately scaled number is interpolated to provide numerical input, the basic unit scale indicator (**u**) may need to be appended to prevent an additional inappropriate default scaling. The number, *N*, may be specified in decimal-fraction form but the parameter finally stored is rounded to an integer number of basic units.

The absolute position indicator (|) may be prepended to a number *N* to generate the distance to the vertical or horizontal place *N*.

- For vertically oriented requests and functions, |*N* becomes the distance in basic units from the current vertical place on the page or in a diversion 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.

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).

Notation

Numerical parameters are indicated in this manual in two ways. A ±*N* means that the argument may take the forms *N*, +*N*, or −*N* and that the corresponding effect is to set the affected parameter to *N*, to increment it by *N*, or to decrement it by *N*, respectively. Plain *N* means that an initial algebraic sign is not an increment indicator but merely the sign of *N*. Generally, unreasonable numerical input is either ignored or truncated to a reasonable value. For example, most requests expect to set parameters to non-negative values; exceptions are **.sp**, **.wh**, **.ch**, **.nr**, and **.if**. 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.

Font and Character Size Control

Fonts

Default mounted fonts are Times Roman (**R**), Times Italic (**I**), Times Bold (**B**), and Special Mathematical (**S**) on physical typesetter positions 1, 2, 3, and 4, respectively. These font styles are shown in Figure 1. The current font, initially Times Roman, may be changed (among the mounted fonts) by use of the **.ft** request or by imbedding at any desired point either **\f***x*, **\f(***xx*, or **\f***N* where *x* and *xx* are the name of a mounted font and *N* is a numerical font position. It is not necessary to change to the Special Font; characters on that font are automatically handled. A request for a named but not mounted font is ignored.

The **troff** processor can be informed that any particular font is mounted by use of the **.fp** request. The list of known fonts is installation dependent. In the subsequent discussion of font-related requests, *F* represents either a 1- or 2-character font name or the numerical font position, 1 through 4. The current font is available as numerical position in the read-only number register **.f**.

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

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 Table D. The three ASCII character exceptions are mapped as follows:

ASCII INPUT		*PRINTED BY troff*	
CHARACTER	*NAME*	*CHARACTER*	*NAME*
′	acute accent	’	close quote
§	grave accent	‘	open quote
—	minus	-	hyphen

The characters ′, §, and − may be input by \\’, \\‘, and \\−, respectively, or by their names. The ASCII characters @, #, ", ′, §, <, >, \\, {, }, ˜, ˆ, and _ exist on the Special Mathematical font and are printed as a one em space if that font is not mounted.

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

- ASCII characters
- Additional characters as may be available on the output device
- Such characters as may be able to be constructed by overstriking or other combinations

• Those characters that can reasonably be mapped into other printable characters.

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

Character Size

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. The **.ps** request is used to change or restore the point size. Alternatively, the point size may be changed between any two characters by imbedding a \s*N* at the desired point to set the size to *N* or a \s±*N* ($1 \leqslant N \leqslant 9$) to increment/decrement the size by *N*; \s**0** restores the previous size. Requested point size values that are between two valid sizes yield the larger of the two. The current size is available in the **.s** number register. The **nroff** formatter ignores type size control. Table E is a summary and explanation of character size requests.

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). A pseudo-page transition onto the first page occurs either when the first break occurs or when the first nondiverted text processing occurs. Arrangements for a trap to occur at the top of the first page must be completed before this transition. A summary and explanation of page control requests is shown in Table F. References to the current diversion mean that the mechanism being described works during both ordinary and diverted output (the former is considered as the top diversion level).

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

Text Filling, Adjusting, and Centering

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 imbedded 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.

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.

Table G is a summary and explanation of filling, adjusting, and centering requests.

Vertical Spacing

Base-line Spacing

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

Extra Line Space

If a word contains a vertically tall construct requiring the output line containing it to have extra vertical space before and/or after it, the *extra line space* function **\x**'*N*' can be imbedded in or attached to that word. In this and other functions having a pair of delimiters around their parameter, the delimiter choice is arbitrary except that it can not 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.

Blocks of Vertical Space

A block of vertical space is ordinarily requested using **.sp**, which honors the no-space mode and which does not space past a trap. A contiguous block of vertical space may be reserved using the **.sv** request.

Table H is a summary and explanation of vertical spacing requests.

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**. Table I is a summary and explanation of line length and indenting requests.

Macros, Strings, Diversions, and Position Traps

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.

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 interpolatedn.

• Arguments indicated by \**$** are interpolated.

• Concealed newline characters indicated by \\<**newline**> are eliminated.

• Comments indicated by \\" are eliminated.

• \t and \a are interpreted as ASCII horizontal tab and start of heading (SOH).

• \\\\ is interpreted as "\\".

• \\. is interpreted as ".".

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

Arguments

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

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

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

and called by

```
.xx Monday 14th
```

to produce the text

```
Today is Monday the 14th.
```

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

• No arguments are available at the top (nonmacro) level in this implementation.

• 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.

Diversions

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

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

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

Traps

Three types of trap mechanisms are available:

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

Macro-invocation traps may be planted using **.wh** requests at any page position including the top. This trap position may be changed using the **.ch** request. Trap positions at or below the bottom of the page have no effect unless or until moved to within the page or rendered effective by an increase in page length. Two traps may be planted at the same position only by first planting them at different positions and then moving one of the traps; the first planted trap will conceal the second unless and until the first one is moved. If the first planted trap is moved back, it again conceals the second trap. The macro associated with a page trap is automatically invoked when a line of text is output whose vertical size reaches or sweeps past the trap position. Reaching the bottom of a page springs the top-of-page trap, if any, provided there is a next page. The distance to the next trap position is available in the **.t** register; if there are no traps between the current position and the bottom of the page, the distance returned is the distance to the page bottom.

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

Table J is a summary and explanation of macros, strings, diversion, and position traps requests.

Number Registers

A variety of predefined number registers (Table K) 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 (Table L), 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

SEQUENCE	*EFFECT ON REGISTER*	*VALUE INTERPOLATED*
\n*x*	none	*N*
\n(*xx*	none	*N*
\n+*x*	*x* incremented by *M*	*N*+*M*
\n−*x*	*x* decremented by *M*	*N*−*M*
\n+(*xx*	*xx* incremented by *M*	*N*+*M*
\n−(*xx*	*xx* decremented by *M*	*N*−*M*

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

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

Table M is a summary and explanation of number registers requests.

Tabs, Leaders, and Fields

Tabs and Leaders

The ASCII horizontal tab character and the ASCII SOH character (the leader) can both be used to generate either horizontal motion or a string of repeated characters. The length of the generated entity is governed by internal tab stops specified with a **.ta** request. The default difference is that tabs generate motion and leaders generate a string of periods; **.tc** and **.lc** offer the choice of repeated character or motion. There are three types of internal tab stops: left justified, right justified, and centered. In the following table:

- *next-string* consists of the input characters following the tab (or leader) up to the next tab (or leader) or end of line

Tabs, Leaders, and Fields

Tabs and Leaders

The ASCII horizontal tab character and the ASCII SOH character (the leader) can both be used to generate either horizontal motion or a string of repeated characters. The length of the generated entity is governed by internal tab stops specified with a **.ta** request. The default difference is that tabs generate motion and leaders generate a string of periods; **.tc** and **.lc** offer the choice of repeated character or motion. There are three types of internal tab stops: left justified, right justified, and centered. In the following table:

• *next-string* consists of the input characters following the tab (or leader) up to the next tab (or leader) or end of line

• *D* is the distance from the current position on the input line (where a tab or leader was found) to the next tab stop

• *W* is the width of *next-string*.

TAB TYPE	*LENGTH OF MOTION OR REPEATED CHARACTERS*	*LOCATION OF next-string*
Left	*D*	Following *D*
Right	*D−W*	Right justified within *D*
Centered	*D−W/2*	Centered on right end of *D*

The length of generated motion is allowed to be negative but that of a repeated character string cannot be. Repeated character strings contain an integer number of characters, and any residual distance is prepended as motion. Tabs (or leaders) found after the last tab stop are ignored, but they may be used as *next-string* terminators.

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

Fields

A field is contained between a pair of field delimiter characters. It consists of substrings separated by padding indicator characters. The field length is the distance on the input line from the position where the field begins to the next tab stop. The difference between the total length of all the substrings and the field length is incorporated as horizontal padding space that is divided among the indicated padding places. The incorporated padding is allowed to be negative. For example, if the field delimiter is "#" and the padding indicator is "^", then

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

specifies a right-justified string with the string *xxx* centered in the remaining space. Table N is a summary and explanation of tab, leader, and field requests.

Input/Output Conventions and Character Translations

Input Character Translations

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

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

• should not be confused with the ASCII control character ESC of the same name

• can be input with the sequence \\

• can be changed with **.ec**, and all that has been said about the default \ becomes true for the new escape character.

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

FORMATTING FACILITIES

Ligatures

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

Backspacing, Underlining, and Overstriking

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

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

Control Characters

Both the *break* control character (.) and the *no-break* control character (') may be changed, if desired. Such a change must be compatible with the design of any macros used in the span of the change and particularly of any trap-invoked macros. A summary and explanation of the **.cc** and **.c2** control character requests are included in Table O.

Output Translation

One character can be made a stand-in for another character using the **.tr** request. All text processing (e.g., character comparisons) takes place with the input (stand-in) character which appears to have the width of the final character. Graphic translation occurs at the moment of output (including diversion). Included in Table O is a summary and explanation of the output translation request.

Transparent Throughput

An input line beginning with a \! is read in copy mode and transparently output (without the initial \!); the text processor is otherwise unaware of the line's presence. This mechanism may be used to pass control information to a post-processor or to imbed control lines in a macro created by a diversion.

Comments and Concealed Newline Characters

An uncomfortably long input line that must stay one line (e.g., a string definition or no-filled text) can be split into many physical lines by ending all but the last one with the escape character (\). The sequence **<newline>** is ignored except in a comment. Comments may be imbedded at the end of any line by prefacing them with \". The newline character at the end of a comment cannot be concealed. A line beginning with \" will appear as a blank line and behave like **.sp 1**; a comment can be on a line by itself by beginning the line with .\".

Local Horizontal/Vertical Motion and Width Function

Local Motion

The functions **v'***N***'** and **h'***N***'** can be used for local vertical and horizontal motion, respectively. The distance *N* may be negative; the positive directions are *rightward* and *downward.* A local motion is one contained within a line. To avoid unexpected vertical dislocations, it is necessary that the net vertical local motion (within a word in filled text and otherwise within a line) balance to zero. The above and certain other escape sequences providing local motion are summarized and explained in Table P. As an example, **E**^{2} is generated by the sequence **E\v'−.5'\s−4\&2\s0\v'.5'**.

Width Function

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

```
.ti−\w'1.'u
```

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

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

- **0** means that all characters in *string* are short lowercase characters without descenders (like **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.

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**.

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

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 ⊕.

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 (⎕).

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 $\left[\mathbf{E}\right]$.

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

$\underline{\text{underlined words}}$

and

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

The function **\L'***Nc***'** will draw a vertical line consisting of the optional character *c* stacked vertically apart one em (one line in **nroff**), with the first two characters overlapped, if necessary, to form a continuous line. The default character is box rule (**\(br**); the other suitable character is bold vertical (**\(bv**). The line is begun without any initial motion relative to the current base line. A positive *N* specifies a line drawn downward, and a negative *N* specifies a line drawn upward. After the line is drawn, no compensating motions are made; the instantaneous base line is at the end of the line.

The horizontal and vertical line drawing functions may be used in combination to produce large boxes. The zero-width *box-rule* and the one-half em wide *underrule* were designed to form corners when using one em vertical spacings.

For example, the macro

```
.de eb
.sp −1      \"compensate for next automatic base-line spacing
.nf         \"avoid possibly overflowing word buffer
\h'−.5n'\L'|\\nau−1'\l'\\n(.lu+1n\(ul'\L'−|\\nau+1'\l'|0u−.5n\(ul'  \"draw
            box
.fi
..
```

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

Hyphenation

The automatic hyphenation may be switched off and on. When switched on with **.hy**, several variants may be set. A hyphenation indicator character may be imbedded in a word to specify desired hyphenation points or may be prepended to suppress hyphenation. In addition, the user may specify a small exception word list. The default condition of hyphenation is off.

Only words that consist of a central alphabetic string surrounded by nonalphabetic strings (usually null) are considered candidates for automatic hyphenation. Words that were input containing hyphens (minus), em-dashes (\**(em**), or hyphenation indicator characters (such as mother-in-law) are always subject to splitting after those characters whether or not automatic hyphenation is on or off. Table Q is a summary and explanation of hyphenation requests.

Three-Part Titles

The titling function **.tl** provides for automatic placement of three fields at the left, center, and right of a line with a title length specifiable with **.lt**. The **.tl** may be used anywhere and is independent of the normal text collecting process. A common use is in header and footer macros. Table R is a summary and explanation of 3-part title requests.

Output Line Numbering

Automatic sequence numbering of output lines may be requested with **.nm**. When in effect, a 3-digit, Arabic number plus a digit-space is prepended to output text lines. Text lines are offset by four digit-spaces and otherwise retain their line length. A reduction in line length may be desired to keep the right margin aligned with an earlier margin. Blank lines, other vertical spaces, and lines generated by **.tl** are not numbered. Numbering can be temporarily suspended with **.nn** or with a **.nm** followed by a later **.nm +0**. In addition, a line number indent *I* and the number-text separation *S* may be specified in digit-spaces. Further, it can be specified that only those line numbers that are multiples of some number *M* are to be printed (the others will appear as blank number fields). Table S is a summary and explanation of output line numbering requests.

Figure 2 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.

Conditional Acceptance of Input

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

- *c* is a 1-character, built-in condition name.

- ! signifies *not.*

- *N* is a numerical expression.

- *string1* and *string2* are strings delimited by any nonblank, non-numeric character not in the strings.

- *anything* represents what is conditionally accepted.

Built-in condition names are:

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

If condition *c* is true, if number *N* is greater than zero, or if strings compare identically (including motions and character size and font), *anything* is accepted as input. If a "!" precedes the condition, number, or string comparison, the sense of the acceptance is reversed.

Any spaces between the condition and the beginning of *anything* are skipped over. The *anything* can be either a single input line (text, macro, or whatever) or a number of input lines. In the multiline case, the first line must begin with a left delimiter "\{" and the last line must end with a right delimiter "\}".

The request **.ie** (if-else) is identical to **.if** except that the acceptance state is remembered. A subsequent and matching **.el** (else) request then uses the reverse sense of that state. The **.ie** - **.el** pairs may be nested. For example:

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

outputs a title if the page number is even, and

```
        .ie\n%>1\{\
        'sp 0.5i
        .tl 'Page %'''
        'sp|1.2i\}
        .el .sp|2.5i
```

treats page 1 differently from other pages.

Environment Switching

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

Insertions From Standard Input

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

If insertions are to be taken from the terminal keyboard while output is being printed on the terminal, the command line option **−q** will turn off the echoing of keyboard input and prompt only with BEL. The regular input and insertion input cannot simultaneously come from the standard input. As an example, multiple copies of a form letter may be prepared by entering insertions for all copies in one file to be used as the standard input and causing the file containing the letter to reinvoke itself by using the **.nx** request. The process would be ended by a **.ex** request in the insertion file. Table V is a summary and explanation of insertions from the standard input requests.

Input/Output File Switching

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

Miscellaneous

Table X is a summary and explanation of miscellaneous requests.

Output and Error Messages

Output from **.tm**, **.pm**, and prompt from **.rd**, as well as various error messages are written onto the UNIX operating system standard message output. The latter is different from the standard output, when compared to the **nroff** formatted output. By default, both are written onto the user's terminal, but they can be independently redirected.

Various error conditions may occur during the operation of the **nroff** and **troff** formatters. Certain less serious errors having only local impact do not cause processing to terminate. Two examples are:

- *word overflow* - caused by a word that is too large to fit into the word buffer (in fill mode)

- *line overflow* - caused by an output line that grew too large to fit in the line buffer.

In both cases, a message is printed, the offending excess is discarded, and the affected word or line is marked at the point of truncation with a * (in **nroff**) or a ☛ (in **troff**). The philosophy is to continue processing, if possible, on the grounds that output useful for debugging may be produced. If a serious error occurs, processing terminates, and an appropriate message is printed. Examples are the inability to create, read, or write files, and the exceeding of certain internal limits that make future output unlikely to be useful. Table Y is a summary and explanation of output and error messages requests.

Compacted Macros

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

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
        .
        .
        .
```

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

Each macro package must be compacted separately by the **nroff** formatter. Compacted macro packages depend on the particular version of the **nroff** formatter that produced them. Any compacted macro packages must be recompacted when a new version of an **nroff** formatter is installed. If it is discovered that a macro package was produced by a different version than that attempting to read it, the **−c** will be abandoned, and the equivalent **−m** option attempted instead.

Install Compacted Files

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

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

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
```

TROFF TUTORIAL

OVERVIEW

An important rule of 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 too difficult for most people to use effectively.

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

For producing text that may contain mathematics or tables, there are a number of macro packages that define formatting rules and operations for specific styles of documents and reduce the amount of direct contact with the **troff** formatter. In particular, the Memorandum Macros (MM) package provides most of the facilities needed for a wide range of document preparation. There are also packages for viewgraphs and other special applications. These packages are easier to use than the **troff** formatter once the user gets beyond the most trivial operations. They should be considered first.

In the few cases where existing packages do not accomplish the job, the solution is 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. The **troff** formatter described is the C language version running on the UNIX operating system at Murray Hill.

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 formula for the area of a circle is typed as follows:

```
Area = \(*p\fIr\fR\|\s8\u2\d\s0
```

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

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 - 6, 7, 8, 9, 10, 11, 12, 14, 16, 18, 20, 22, 24, 28, and 36. Point size is rounded up to the next valid value, with a maximum of 36, 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** processor begins 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. 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** formatter uses 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 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).

FONTS AND SPECIAL CHARACTERS

The **troff** processor and the typesetter allow 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. The Greek, mathematical symbols, and miscellany of the special font are listed in Table D.

The **troff** 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, although only four can be used at any given time. The **.fp** request tells the **troff** formatter what fonts are actually mounted on the typesetter. For example:

```
.fp 3 H
```

says that the Helvetica font is mounted on position 3. A list of fonts and what they look like are shown in Figure 1. 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.

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 Table D.

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.

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 phototypesetter is 7.54 inches. 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 serman; 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. For example, all paragraphs in this manual effectively begin with the **.ti 3** request. Since no units are specified, the line is indented three ems by default. 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.

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 done 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
700800900
.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
NameAge
```

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.

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

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.

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.

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

```
>\h'−0.3m'>
```

to produce >>.

Frequently, \h is used with the "width function" \w to generate motions equal to the width of some 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 of the same width as a digit.

- The \\<**space**> is an unpaddable character the width of a space.

- The \| is 1/6 the width of a space.

- The \^ is 1/12 the width of a space.

- 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.

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 base line. Thus big brackets are obtained by constructing them with piled-up smaller pieces.

Drawing Lines

A convenient facility for drawing horizontal and vertical lines of arbitrary length with arbitrary characters is provided by the **troff** formatter. A 1-inch long line is printed with a **\l'1i'** sequence. 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 part titled "Table Formatting Program" describes other ways of providing horizontal and vertical lines.

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.

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:

```
.spfR
.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 that start and end a block of offset, unfilled text is

```
.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.

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**) 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 leftover 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.

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**.

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 a number from one to four).

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

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

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

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

```
.de PP
.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) before any arithmetic is done, so **1i/2u** evaluates to **0.5i** correctly.

The scale indicator **u** often has to appear when least expected, in particular when arithmetic is being done in a context that implies horizontal or vertical dimensions. For example, **.ll 7/2i** is not 3½ inches. Instead, it is really 7 ems/2 inches. When translated into machine units, it becomes 0. This is because the default units for horizontal parameters (like **.ll**) are ems. Another incorrect try is **.ll 7i/2**. The 2 is 2 ems, so 7i/2 is small, although not 0. The **.ll 7i/2u** must be used. A safe rule is to attach a scale indicator to every number, even constants.

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

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

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

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.

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.

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.

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.

NROFF/TROFF 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 such as page margins and footnotes are deliberately not built into the **nroff** and **troff** formatters. Instead, the macro and string definition, number register, diversion, environment switching, page-position trap, and conditional input mechanisms provide the basis for user-defined implementations.

Examples in the following text are intended to be useful and somewhat realistic but will not necessarily cover all relevant contingencies. Explicit numerical parameters are used to make the examples easier to read and to illustrate typical values. In many cases, number registers 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.

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
```

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.

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.

FOOTNOTE PROCESSING

The footnote mechanism is used by imbedding the footnotes in the input text at the point of reference demarcated by an initial **.fn** and a terminal **.ef**.

```
.fn
Footnote text and control lines.
.ef
```

The following macro definitions cause footnotes to be processed in a separate environment and diverted for later printing in the space immediately prior to the bottom margin. There is provision for the case where the last collected footnote does not completely fit in the available space:

```
.de hd                     \"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.

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.

TABLE FORMATTING PROGRAM

INTRODUCTION

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

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

USAGE

On the UNIX operating 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, and *ms* or *mm* macro requests, the normal command is

```
tbl file1 file2...|eqn|troff −ms
```

The usual options may be used on the **troff** formatter. Usage of the **nroff** formatter is similar to that of **troff**, but only TELETYPE® Model 37 and Diablo-mechanism (DASI or GSI) terminals can print boxed tables. 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. The only other command-line options recognized by **tbl** are *-ms* and *-mm*. They are turned into commands to fetch the corresponding macro files. It is usually more convenient to place these arguments on the **troff** formatter part of the command line, but they are accepted by **tbl** as well.

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 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 $± 16$
```

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

```
.TS H
```

a line of the form

```
.TH
```

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

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 proceses 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
- A format section describing individual columns and rows of the table
- Data to be printed.

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

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.

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 string lines up where a dot normally would (the \& disappears from the final output). In the following example, items shown in the **INPUT** column will be aligned (in a numerical column) as shown in the **OUTPUT** column.

```
INPUT:            OUTPUT:
.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
```

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. 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 delimeter) 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.

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.

EXAMPLES

Figures 3 through 8 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 7 also indicates the requesting of bold type print in the format area.

MATHEMATICS TYPESETTING PROGRAM

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:

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

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, the **troff** formatter, so mathematical expressions can be embedded in the running text of a manuscript, and the entire document produced in one process. Typical mathematical expressions include size and font changes, positioning, line drawing, and other necessary functions to print according to mathematical conventions, and are done automatically. The syntax of the language is specified by a small context-free grammar; a compiler-compiler is used to make a compiler that translates this language into typesetting commands. Output may be produced on either a phototypesetter or on a terminal with forward and reverse half-line motions.

USAGE

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

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

```
eqn files | troff
```

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

```
eqn files | troff -ms
```

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

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

```
neqn files | nroff
```

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

```
neqn files | nroff -Tx
```

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 should be considered as an initial step in formatting a document.

LANGUAGE

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.

FORMATTING FACILITIES

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 compiler for the language was built using a compiler-compiler.

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.

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.

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.

USER'S GUIDE

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. 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 packages, **−ms** and **−mm**, allow equations to be centered, indented, left-justified, and numbered. The **−ms** package centers (by default) equations. To left-justify an equation, the **.EQ L** macro is used. A **.EQ I** macro will indent the equation. Any of these sequences can be followed by an arbitrary equation number placed at the right margin. For example, the input

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

produces the output

$x=f(y/2)+y/2$

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
```

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.

Spaces and New Lines

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.

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, to get

$x ~=~ y ~+~ z$

the following expression is input

```
x~=~y~+~z
```

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 appears as "f(pi)" in the output. A list of **eqn** names appears in Table Z. 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:

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

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
```

produces

$$x_{i_1}$$

A subscript and superscript on the same thing are printed one above the other if the subscript comes first. The construct "something sub something sup something" is recognized as a special case.

```
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
```

Braces

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

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

```
e sup {i omega t}
```

produces

$$e^{i\omega t}$$

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

Braces can occur within braces if necessary. The statement

```
e sup {i pi sup {rho +1}}
```

generates

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

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

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

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

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

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.

Summations, Integrals, and Similar Constructions

Summations, integrals, and similar constructions are easy.

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

produces

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

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

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

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

```
int
prod
union
inter
```

which become, respectively

$\int$

$\prod$

$\bigcup$

$\bigcap$

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

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

is

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

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

$\mathbf{x}y$

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.

FORMATTING FACILITIES

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

```
A = fat {pi r sup 2}
```

produces

$$A = \boldsymbol{\pi r^2}$$

Legal sizes which may follow *size* are

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

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

```
size +2
```

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

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

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

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

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

Diacritical Marks

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

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

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

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

It is made by typing

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

Quoted Text

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

```
italic "sin(x)" + sin (x)
```

produces

sin(x)+sin(*x*)

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

"{ size alpha } "

prints

{ *size alpha* }

and

roman "{ size alpha }"

prints

{ size alpha }

The "" construction is often used as a place-holder when grammatically **eqn** needs something, but nothing is actually wanted on the output.

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

$$x+y=z$$
$$\phantom{x+{}}x=1$$

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

```
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.

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

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

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

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

```
left "" ... right)
```

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

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

$$sign(x) \equiv \left\{ \begin{array}{rll} 1 & \text{if} & x>0 \\ 0 & \text{if} & x=0 \\ -1 & \text{if} & x<0 \end{array} \right.$$

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.

Matrices

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

$$\begin{array}{ll} x_i & x^2 \\ y_i & y^2 \end{array}$$

the following 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.

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 in the middle of text lines can be typed.

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

The three lines added to the beginning of the document set both the left and right 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 $\geqslant 0$.

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.

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. It is possible to redefine keywords like *sup*, *sub*, or *over*. 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. By making / mean *over* with the following statement:

```
define / ' over '
```

or by redefining *over* as / with:

```
define over ' / '
```

the keyword is redefined.

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

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.

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.

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 ommitted 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.

FORMATTING FACILITIES EXAMPLES

The following 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.

Figure 1 — Font Style Examples

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 2 — Example of Output Line Numbering

Automatic sequence numbering of output lines may be requested with **.nm**. When in effect, a 3-digit, arabic number plus a digit-space is prepended to output text lines. Text lines are offset by four digit-spaces and otherwise retain their line length. A reduction in line length may be desired to keep the right margin aligned with an earlier margin. Blank lines, other vertical spaces, and lines generated by **.tl** are not numbered. Numbering can be temporarily suspended with **.nn** or with a **.nm** followed by a later **.nm +0**. In addition, a line number indent *I* and the number-text separation *S* may be specified in digit-spaces. Further, it can be specified that only those line numbers that are multiples of some number *M* are to be printed (the others will appear as blank number fields). Table S 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.

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 serman; I find I was wrong; for on returning home and referring to the book whence I thought it was taken, I found it there."

Figure 3 — Table Using "box" Option

INPUT:

```
.TS
box;
c c c
l l l.
LanguageAuthorsRuns on
.sp
FortranManyAlmost anything
Pl/1IBM360/370
CBTL11/45,H6000,370
BLISSCarnegie-MellonPDP-10,11
IDSHoneywellH6000
PascalStanford370
.TE
```

OUTPUT:

Language	Authors	Runs on
Fortran	Many	Almost anything
PL/1	IBM	360/370
C	BTL	11/45,H6000,370
BLISS	Carnegie-Mellon	PDP-10,11
IDS	Honeywell	H6000
Pascal	Stanford	370

Figure 4 — Table Using "allbox" Option

INPUT:

```
.TS
allbox;
c s s
c c c
n n n.
AT&T Common Stock
YearPriceDividend
197141-54$2.60
241-542.70
346-552.87
440-533.24
545-523.40
651-59.95*
.TE
* (first quarter only)
```

OUTPUT:

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

* (first quarter only)

Figure 5 — Table Using "vertical bar" Key Letter Feature

INPUT:

```
.TS
box;
c s s
c| c| c
l| l| n.
Major New York Bridges
_
BridgeDesignerLength
_
BrooklynJ. A. Roebling1595
ManhattanG. Lindenthal1470
WilliamsburgL. L. Buck1600
_
QueensboroughPalmer &1182
  Hornbostel
_
1380
TriboroughO. H. Ammann_
383
_
Bronx WhitestoneO. H. Ammann2300
Throgs NeckO. H. Ammann1800
.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 6 — Table Using Horizontal Lines In Place Of Key Letters

INPUT:

```
.TS
box;
L L L
L L _
L L |LB
L L _
L L L.
januaryfebruarymarch
aprilmay
junejulyMonths
augustseptember
octobernovemberdecember
.TE
```

OUTPUT:

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

Figure 7 — Table Using Additional Command Lines

INPUT:

```
.TS
box;
cfB s s s.
Composition of Foods
_
.T&
c | c s s
c | c s s
c | c | c | c.
FoodPercent by Weight
\_
\ProteinFatCarbo-
\\\hydrate
_
.T&
l | n | n | n.
Apples.4.513.0
Halibut18.45.2...
Lima beans7.5.822.0
Milk3.34.05.0
Mushrooms3.5.46.0
Rye bread9.0.652.7
.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 8 — Table Using Text Blocks

INPUT:

```
.TS
allbox;
cfI s s
c cw(1.5i) cw(1.5i)
l l l.
New York Area Rocks
.sp
EraFormationAge (years)
PrecambrianReading Prong>1 billion
PaleozoicManhattan Prong400 million
MesozoicT{
.na
Newark Basin, incl.
Stockton,Lockatong, and Brunswick
formations
.ad
T}200 million
CenozoicCoastal PlainT{
.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

Table A

CROSS REFERENCE
REQUEST NAME TO TABLE NUMBER

NAME	*NUMBER*	*NAME*	*NUMBER*
ab	*Y*	*ls*	*H*
ad	G	lt	R
af	M	mc	X
am	J	mk	F
as	J	na	G
bd	C	ne	F
bp	F	nf	G
br	G	nh	Q
c2	O	nm	S
cc	O	nn	S
ce	G	nr	M
ch	J	ns	H
co	X	nx	W
cs	E	os	H
cu	O	pc	R
da	J	pi	W
de	J	pl	F
di	J	pm	X
ds	J	pn	F
dt	J	po	F
ec	O	ps	E
el	T	rd	V
em	J	rm	J
eo	O	rn	J
ev	U	rr	M
ex	V	rs	H
fc	N	rt	F
fi	G	so	W
fl	X	sp	H
fp	C	ss	E
ft	C	sv	H
hc	Q	ta	N
hw	Q	tc	N
hy	Q	ti	I

CROSS REFERENCE
REQUEST NAME TO TABLE NUMBER

NAME	*NUMBER*	*NAME*	*NUMBER*
ie	***T***	***tl***	***R***
if	T	tm	X
ig	X	tr	O
in	I	uf	O
it	J	ul	O
lc	N	vs	H
lg	O	wh	J
ll	I		

Table B

ESCAPE SEQUENCES FOR CHARACTERS, INDICATORS, AND FUNCTIONS

ESCAPE SEQUENCE	*MEANING*
\\	\ (to prevent or delay the interpretation of \)
\′	Acute accent
\§	Grave accent
\−	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 **nroff**)
\ˆ	1/12 em half-narrow space character (zero width **nroff**)
\&	Nonprinting zero width character
\!	Transparent line indicator
\"	Beginning of comment
\$N	Interpolate argument (1 ⩽ N ⩽ 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 **nroff**)
\e	Printable version of current escape character
\fx,\f(xx,\fN	Change to font named x or xx or position N
\gx,\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
\jx,\j(xx	Mark the current horizontal output position in register x or xx
\kx	Mark horizontal input place in register x
\l'Nc'	Horizontal line drawing function (optionally with c)

ESCAPE SEQUENCES FOR CHARACTERS, INDICATORS, AND FUNCTIONS

ESCAPE SEQUENCE	*MEANING*
\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 **nroff**)
\s*N*,\s±*N*	Point-size change function
\t	Noninterpreted horizontal tab
\u	Reverse (up) ½ em vertical motion (½ line **nroff**)
\v'*N*'	Local vertical motion
\w'*string*'	Interpolate width of *string*
\x'*N*'	Extra line-space function (negative before, positive after)
\z*c*	Print *c* with zero width (without spacing)
\X	Any character not listed above

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

Table C

FONT CONTROL REQUESTS

REQUEST FORM	*INITIAL VALUE*	*IF NO ARGUMENT*	*EXPLANATION*
.bd *F N*	off	-	Embolden font *F* by *N*−1 units. Characters in font *F* will be artificially emboldened by printing each one twice, separated by *N*−1 basic units. A reasonable value for *N* is 3 when the character size is in the vicinity of 10 points. If *N* is missing, the embolden mode is turned off. The mode must still (or again) be in effect when the characters are physically printed. There is no effect in the **nroff** formatter.
.bd *S F N*	off	-	Embolden special font when current font is *F*. The characters in the special font will be emboldened whenever the current font is *F*. The mode must still (or again) be in effect when the characters are physically printed. There is no effect in the **nroff** formatter.
.fp *N F*	R,I,B,S	ignored	Font position. A font named *F* is mounted on position *N* (1 through 4). It is a fatal error if *F* is not known. The phototypesetter has four fonts physically mounted. Each font consists of a film strip which can be mounted on a numbered quadrant of a wheel. (Cont'd)

FONT CONTROL REQUESTS

REQUEST FORM	*INITIAL VALUE*	*IF NO ARGUMENT*	*EXPLANATION*
			The default mounting sequence assumed by the **troff** formatter is R, I, B, and S on positions 1, 2, 3, and 4.
.ft *F*	Roman	previous	Change to font *F* (*F* is *x*, *xx*, 1 through 4, or *P*). Font *P* means the previous font. For font changes within a line of text, sequences **\f***x*, **\f(***xx*, or **\f***N* can be used. Relevant parameters are a part of the current environment.

Table D

NAMING CONVENTIONS FOR NON-ASCII CHARACTERS

Non-ASCII characters and *minus* on the standard fonts.

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

NAMING CONVENTION FOR NON-ASCII CHRACTERS

Non-ASCII characters and ´, §, _, +, −, =, and ∗ on the special font.

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

NAMING CONVENTION FOR NON-ASCII CHRACTERS

Non-ASCII characters and ′, §, _, +, −, =, and ∗ on the special font.

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

NAMING CONVENTION FOR NON-ASCII CHRACTERS

Non-ASCII characters and ′, §, _, +, −, =, and ∗ on the special font.

CHARACTER	*INPUT*	*CHARACTER*
	NAME	*NAME*
≠	\(!=	not equal
→	\(−>	right arrow
←	\(<−	left arrow
↑	\(ua	up arrow
↓	\(da	down arrow
×	\×	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
	\(bs	Bell logo
|	\(or	or
○	\(ci	circle
⎧	\(lt	left top (big brace)
⎩	\(lb	left bottom (big brace)

NAMING CONVENTION FOR NON-ASCII CHRACTERS

Non-ASCII characters and ′, §, _, +, −, =, and ∗ on the special font.

CHARACTER	*INPUT NAME*	*CHARACTER NAME*
⎫	\\(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)

Table E

CHARACTER SIZE CONTROL REQUESTS			
REQUEST FORM	*INITIAL VALUE*	*IF NO ARGUMENT*	*EXPLANATION*
.cs *F N M*	off		Set constant character space (width) mode on for font *F* (if mounted). The width of every character is assumed to be *N*/36 ems. If *M* is absent, the em is that of the character point size; if *M* is given, the em is *M*-points. All affected characters are centered in this space including those with an actual width larger than this space. Special font characters occurring while the current font is *F* are also so treated. If *N* is absent, the mode is turned off. The mode must still (or again) be in effect when the characters are printed. There is no effect in the **nroff** formatter.
.ps ±*N*	10 point	previous	Set point size to ±*N*. Any valid positive size value may be requested; if invalid, the next larger valid size will result (maximum of 36). Valid point sizes are: 6, 7, 8, 9, 10, 11, 12, 14, 16, 18, 20, 22, 24, and 36. A paired sequence +*N*, −*N* will work because the previous requested value is remembered. For point size changes within a line of text, sequences \s*N* or \s±*N* can be used. Relevant parameters are a part of the current environment. There is no effect in the **nroff** formatter.

CHARACTER SIZE CONTROL REQUESTS			
REQUEST FORM	*INITIAL VALUE*	*IF NO ARGUMENT*	*EXPLANATION*
.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.

Table F

PAGE CONTROL REQUESTS

REQUEST FORM	*INITIAL VALUE**	*IF NO ARGUMENT*	*EXPLANATION*
.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.
.mk *R*	none	internal	Mark current vertical place in an internal register (associated with the current diversion level) or in register *R*, if given. The request is used in conjunction with "return to marked vertical place in current diversion" request (**.rt**). Mode or relevant parameters are associated with current diversion level.
.ne *N*	-	$N = 1V$	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. (Cont'd)

PAGE CONTROL REQUESTS

REQUEST FORM	*INITIAL VALUE**	*IF NO ARGUMENT*	*EXPLANATION*
			If *D* is less than vertical spacing (*V*), another line could still be output and spring the trap. In a diversion, *D* is the distance to the diversion trap (if any) or is very large. Mode or relevant parameters are associated with current diversion level.
.pl ±*N*	11in	11in	Page length set to ±*N*. The internal limitation is about 75 inches in the **troff** formatter and 136 inches in the **nroff** formatter. Current page length is available in the **.p** register. The scale indicator is ignored if not specified in the request.
.pn ±*N*	*N* = 1	ignored	Page number. The next page (when it occurs) will have the page number ±*N*. The request must occur before the initial pseudopage transition to affect the page number of the first page. The current page number is in the % register.
.po ±*N*	0; 26/27in	previous	Page offset. The current left margin is set to ±*N*. The scale indicator is ignored if not specified in the request. The **troff** formatter initial value provides about 1 inch of paper margin including the physical typesetter margin of 1/27 inch. (Cont'd)

PAGE CONTROL REQUESTS

REQUEST FORM	*INITIAL VALUE**	*IF NO ARGUMENT*	*EXPLANATION*
			In the **troff** formatter the maximum (line-length) + (page-offset) is about 7.54 inches. The current page offset is available in the **.o** register.
.rt ±*N*	none	internal	Return (upward only) to marked vertical place in current diversion. If ±*N* (with respect to place) is given, the vertical place is ±*N* from the top of the page or diversion. If *N* is absent, the vertical place is marked by a previous **.mk**. The **.sp** request may be used in all cases instead of **.rt** by spacing to the absolute place stored in an explicit register; e.g., using the sequence **.mk** *R*....**sp** *R***u** Mode or relevant parameters are associated with current diversion level. The scale indicator is ignored if not specified in the request.

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

Table G

TEXT FILLING, ADJUSTING, AND CENTERING REQUESTS

REQUEST FORM	*INITIAL VALUE*	*IF NO ARGUMENT*	*EXPLANATION*
.ad *N*	adjust	adjust	Adjust. Output lines are adjusted with mode *N*. If the type indicator (*N*) is present, the adjustment type is as follows: *N ADJUSTMENT TYPE* l adjust left margin only r adjust right margin only c center b or n adjust both margins absent unchanged The adjustment type indicator *N* may also be a number obtained from the **.j** register. If fill mode is not on, adjustment will be deferred. Relevant parameters are a part of the current environment.
.br	-	-	Break. Filling of the line currently being collected is stopped and the line is output without adjustment. Text lines beginning with space characters and empty text lines (blank lines) also cause a break.
.ce *N*	off	$N = 1$	Center. The next *N* input text lines are centered within the current line-length. If $N=0$, any residual count is cleared. A break occurs after each of the *N* input lines. If the input line is too long, it will be left adjusted. The request normally causes a break. Relevant parameters are a part of the current environment.
.fi	fill	-	Fill mode. The request causes a break. Subsequent output lines are filled to provide an even right margin. Relevant parameters are a part of the current environment.

TEXT FILLING, ADJUSTING, AND CENTERING REQUESTS

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

Table H

VERTICAL SPACING REQUESTS

REQUEST FORM	*INITIAL VALUE**	*IF NO ARGUMENT*	*EXPLANATION*
.ls *N*	$N = 1$	previous	Line spacing set to $\pm N$. Output $N-1$ blank lines (*V*s) after each output text line. If the text or previous appended blank line reached a trap position, appended blank lines are omitted. Relevant parameters are a part of the current environment.
.ns	space	-	Set no-space mode on. The no-space mode inhibits **.sp** and **.bp** requests without a next page number. It is turned off when a line of output occurs or with the **.rs** request. Mode or relevant parameters are associated with current diversion level.
.os	-	-	Output saved vertical space. The request is used to output a block of vertical space requested by an earlier **.sv** request. The no-space mode (**.ns**) has no effect.
.rs	-	-	Restore spacing. The no-space mode (**.ns**) is turned off. Mode or relevant parameters are associated with current diversion level.
.sp *N*	-	$N = 1V$	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.

VERTICAL SPACING REQUESTS

REQUEST FORM	*INITIAL VALUE**	*IF NO ARGUMENT*	*EXPLANATION*
.sv *N*	-	$N = 1V$	Save a contiguous vertical block of size *N*. If the distance to the next trap is greater than *N*, *N* vertical spaces are output. If the distance to the next trap is less than *N*, no vertical space is immediately output; but *N* is remembered for later output (**.os**). Subsequent **.sv** requests overwrite any still remembered *N*. The no-space mode (**.ns**) has no effect. The scale indicator is ignored if not specified in the request.
.vs *N*	1/6in; 12pts	previous	Set vertical base-line spacing size *V*. Transient extra vertical spaces are available with \x'*N*'. The scale indicator is ignored if not specified in the request. Relevant parameters are a part of the current environment.
Blank line	-	-	This condition causes a break and output of a blank line (just as does **.sp 1**).

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

Table I

LINE LENGTH AND INDENTING REQUESTS

REQUEST FORM	*INITIAL VALUE*	*IF NO ARGUMENT*	*EXPLANATION*
.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.
.ll $\pm N$	6.5 in	previous	Line length. The line length is set to $\pm N$. In the **troff** formatter, the maximum (line-length) + (page-offset) is about 7.54 inches. The scale indicator is ignored if not specified in the request. Relevant parameters are a part of the current environment.
.ti $\pm N$	-	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.

Table J

MACROS, STRINGS, DIVERSIONS, AND POSITION TRAPS REQUESTS

REQUEST FORM	*INITIAL VALUE*	*IF NO ARGUMENT*	*EXPLANATION*
.am	*xx yy*	*.yy*=..	Append to macro *xx* (append version of **.de**).
.as *xx string*		ignored	Append *string* to string *xx* (append version of **.ds**).
.ch *xx N*	-	-	Change trap location. Change the position for macro *xx* to be *N*. In the absence of *N*, the trap is removed. The scale indicator is ignored if not specified.
.da *xx*	-	end	Divert and append to macro *xx*. Mode or relevant parameters are associated with current diversion level.
.de *xx yy*	.	*yy*=..	Define or redefine macro *xx*. Contents of the macro begin on the next line. Lines are copied in copy mode until the definition is terminated by a line beginning with *.yy*. *yy* is then called. In the absence of *yy*, definition is terminated by a line beginning with "..". A macro contains **.de** requests provided terminating macros differ or contained definition terminator is concealed; ".." can be concealed as "\\.." which copies as "\.." and rereads as "..".

MACROS, STRINGS, DIVERSIONS, AND POSITION TRAPS REQUESTS

REQUEST FORM	*INITIAL VALUE*	*IF NO ARGUMENT*	*EXPLANATION*
.di *xx*	-	end	Divert output to macro *xx*. Normal text processing occurs during diversion except that page offsetting is not done. The diversion ends when the request **.di** or **.da** is encountered without an argument; extraneous requests of this type should not appear when nested diversions are being used. Mode or relevant parameters are associated with current diversion level.
.ds *xx string*		ignored	Define a string *xx* containing *string*. Any initial double-quote in *string* is stripped to permit initial blanks.
.dt *N xx*	-	off	Install a diversion trap at position *N* in the current diversion to invoke macro *xx*. Another **.dt** will redefine the diversion trap. If no arguments are given, the diversion trap is removed. Mode or relevant parameters are associated with current diversion level. The scale indicator is ignored if not specified in the request.
.em *xx*	none	none	End macro. Macro *xx* will be invoked when all input has ended. the end of the last file processed.

MACROS, STRINGS, DIVERSIONS, AND POSITION TRAPS REQUESTS

REQUEST FORM	*INITIAL VALUE*	*IF NO ARGUMENT*	*EXPLANATION*
.it *N xx*	-	off	Input-line-count trap. An input-line-count trap is set to invoke the macro *xx* after *N* lines of text input have been read (control or request lines do not count). Text may be in-line or interpolated by in-line or trap-invoked macros. Relevant parameters are a part of the current environment.
.rm *xx*	-	ignored	Remove. A request, macro, or string is removed. The name *xx* is removed from the name list and any related storage space is freed. Subsequent references have no effect.
.rn *xx yy*	-	ignored	Rename. Rename request, macro, or string from *xx* to *yy*. If *yy* exists, it is first removed.
.wh *N xx*	-	-	When. A location trap is set to invoke macro *xx* at page position *N*; a negative *N* is interpreted with respect to the page bottom. Any macro previously planted at *N* is replaced by *xx*. A zero *N* refers to the top of a page. In the absence of *xx*, the first found trap at *N*, if any, is removed. The scale indicator is ignored if not specified in the request.

Table K

PREDEFINED GENERAL NUMBER REGISTERS

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

Table L

PREDEFINED READ-ONLY NUMBER REGISTERS

REGISTER NAME	*DESCRIPTION*
.$	Number of arguments available at the current macro level.
.A	Set to **1** in the **troff** formatter if **-a** option used; always **1** in the **nroff** formatter.
.F	Value is a *string* that is the name of the current input file.
.H	Available horizontal resolution in basic units.
.L	Contains the current line spacing parameter (the value of the most recent **.ls** request).
.P	appear in the **−o** option list.
.T	Set to **1** in the **nroff** formatter if **-T** option used; always **0** in the **troff** formatter.
.V	Available vertical resolution in basic units.
.a	Post-line extra line space most recently utilized using **x**′*N*′.
.c	Number of lines read from current input file.
.d	Current vertical place in current diversion; equal to **nl** if no diversion.
.f	Current font as physical quadrant (1 through 4).
.h	Text base-line high-water mark on current page or diversion.
.i	Current indent.
.j	Indicates the current adjustment mode and type. Can be saved and later given to the **.ad** request to restore a previous mode.
.k	Contains the horizontal size of the text portion (without indent) of the current partially collected output line, if any, in the current environment.
.l	Current line length.
.n	Length of text portion on previous output line.
.o	Current page offset.
.p	Current page length.
.s	Current point size.
.t	Distance to the next trap.

PREDEFINED READ-ONLY NUMBER REGISTERS

REGISTER NAME	*DESCRIPTION*
.u	Equal to **1** in fill mode and **0** in no-fill mode.
.v	Current vertical line spacing.
.w	Width of previous character.
.x	Reserved version-dependent register.
.y	Reserved version-dependent register.
.z	Name of current diversion.

Table M

NUMBER REGISTERS REQUESTS

REQUEST FORM	*INITIAL VALUE*	*IF NO ARGUMENT*	*EXPLANATION*
.af *R c*	Arabic	-	Assign format. Format *c* is assigned to register *R*. Available formats are: *c – NUMBERING SEQUENCE* 1 – 0,1,2,3,4,5,... 001 – 000,001,002,003,004,005,... i – 0,i,ii,iii,iv,v,... I – 0,I,II,III,IV,V,... a – 0,a,b,...,z,aa,ab,...,zz,aaa,... A – 0,A,B,...,Z,AA,AB,...,ZZ,AAA,... An Arabic format having *N* digits specifies a field width of *N* digits. Read-only registers and width function are always arabic.
.nr *R* ±*N M*	-	-	Number register. The number register *R* is assigned the value ±*N* with respect to the previous value, if any. The automatic incrementing value is set to *M*. The number register value (*N*) is ignored if not specified in the request.
.rr *R*	-	-	Remove register. The number register *R* is removed. If many registers are being created dynamically, it may be necessary to remove registers that are no longer used in order to recapture internal storage space for newer registers.

Table N

TABS, LEADERS, AND FIELDS REQUESTS

REQUEST FORM	*INITIAL VALUE**	*IF NO ARGUMENT*	*EXPLANATION*
.fc *a b*	off	off	Field delimiter is set to *a*. The padding indicator is set to the space character or to *b*, if given. In the absence of arguments, the field mechanism is turned off.
.lc *c*	.	none	Leader repetition character becomes *c* or is removed specifying motion. Relevant parameters are a part of the current environment.
.ta *Nt*...	8n; 0.5 in	none	Set tab stops and types. The adjustment within the tab is as follows: *t – ADJUSTMENT TYPE* R – right C – centering absent – left Tab stops for the **troff** formatter are preset every 0.5 inch; Tab stops for the **nroff** formatter are preset every eight nominal character widths. Stop values are separated by spaces, and a value preceded by + is treated as an increment to the previous stop value. Relevant parameters are a part of the current environment. The scale indicator is ignored if not specified in the request.

TABS, LEADERS, AND FIELDS REQUESTS

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

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

Table O

INPUT AND OUTPUT CONVENTIONS AND CHARACTER TRANSLATIONS REQUESTS

REQUEST FORM	*INITIAL VALUE**	*IF NO ARGUMENT*	*EXPLANATION*
.cc *c*	.	.	Set control character to **c** or reset to ".". Relevant parameters are a part of the current environment.
.cu *N*	off	*N* = 1	Continuous underline in the **nroff** formatter. A variant of **.ul** that causes every character to be underlined. Identical to **.ul** in the **troff** formatter. Relevant parameters are a part of the current environment.
.c2 *c*	'	'	Set no-break control character to **c** or reset to "'". Relevant parameters are a part of the current environment.
.ec *c*	\	\	Set escape character to \ or to *c* if given.
.eo	on	-	Turn escape character mechanism off.
.lg *N*	off;on	on	Ligature mode is turned on if *N* is absent or nonzero and turned off if *N*=0. If *N*=2, only the 2-character ligatures are automatically invoked. Ligature mode is inhibited for request, macro, string, register, file names, and copy mode. There is no effect in the **nroff** formatter.
.tr *abcd*...	none	-	Translate *a* into *b*, *c* into *d*, etc. on output. If an odd number of characters is given, the last one will be mapped into the space character. To be consistent, a particular translation must stay in effect from input to output time.

INPUT AND OUTPUT CONVENTIONS AND CHARACTER TRANSLATIONS REQUESTS

REQUEST FORM	*INITIAL VALUE**	*IF NO ARGUMENT*	*EXPLANATION*
.uf *F*	Italic	Italic	Underline font set to *F* (to be switched to by **.ul**). In the **nroff** formatter *F* may not be on position 1 (initially Times Roman).
.ul *N*	off	$N = 1$	Underline in the **nroff** formatter (italicize in **troff**) the next *N* input text lines. Switch to underline font saving the current font for later restoration; other font changes within the span of a **.ul** will take effect, but the restoration will undo the last change. Output generated by **.tl** is affected by the font change but does not decrement *N*. If *N* is greater than 1, there is the risk that a trap interpolated macro may provide text lines within the span, which environment switching can prevent. Relevant parameters are a part of the current environment.

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

Table P

LOCAL MOTIONS

VERTICAL LOCAL MOTION

FUNCTION	*EFFECT IN*	
	troff	*nroff*
\v'*N*'	Move distance *N*	
\u	½ em up	½ line up
\d	½ em down	½ line down
\r	1 em up	1 line up

HORIZONTAL LOCAL MOTION

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

Table Q

HYPHENATION REQUESTS

REQUEST FORM	*INITIAL VALUE*	*IF NO ARGUMENT*	*EXPLANATION*
.hc *c*	\%	\%	Hyphenation character. Hyphenation indicator character is set to *c* or to the default "\%". The indicator does not appear in the output. Relevant parameters are a part of the current environment.
.hw *word1*...	-	ignored	Exception words. Hyphenation points in words are specified with imbedded minus signs. Versions of a word with terminal s are implied; i.e., *dig-it* implies *dig−its*. This list is examined initially and after each suffix stripping. Space available is small — about 128 characters.
.hy *N*	off,$N=0$	on,$N=1$	Hyphenate. Automatic hyphenation is turned on for $N \geqslant 1$ 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.

HYPHENATION REQUESTS

REQUEST FORM	*INITIAL VALUE*	*IF NO ARGUMENT*	*EXPLANATION*
			For $N=8$ the first two characters of a word are not divided. These values are additive; i.e., $N=14$ invokes all three restrictions. Relevant parameters are a part of the current environment.
.nh	no hyphen	-	No hyphenation. Automatic hyphenation is turned off. Relevant parameters are a part of the current environment.

Table R

THREE-PART TITLES REQUESTS

REQUEST FORM	*INITIAL VALUE*	*IF NO ARGUMENT*	*EXPLANATION*
.lt $\pm N$	6.5 in	previous	Length of title set to $\pm N$. Line length and title length are independent. Indents do not apply to titles; page offsets do. Relevant parameters are a part of the current environment. The scale indicator is ignored if not specified in the request.
.pc *c*	%	off	Page number character set to *c* or removed. The page number register remains %.
.tl'*left*'*center*'*right*'			Three-part title. The strings *left*, *center*, and *right* are respectively left-adjusted, centered, and right-adjusted in the current title length. Any of the strings may be empty, and overlapping is permitted. If the page number character (initially %) is found within any of the fields, it is replaced by the current page number having the format assigned to register %. Any character may be used as the string delimiter.

Table S

OUTPUT LINE NUMBERING REQUESTS

REQUEST FORM	*INITIAL VALUE*	*IF NO ARGUMENT*	*EXPLANATION*
.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.
.nn N	-	$N=1$	Next N lines are not numbered. Relevant parameters are a part of the current environment.

Table T

CONDITIONAL ACCEPTANCE OF INPUT REQUESTS

REQUEST FORM	*INITIAL VALUE*	*IF NO ARGUMENT*	*EXPLANATION*
.el *anything*		-	The "else" portion of "if-else".
.ie *c anything*		-	The "if" portion of "if-else". The *c* can be any of the forms acceptable with the **.if** request.
.if *c anything*		-	If condition *c* true, accept *anything* as input; for multiline case, use \\{*anything*\\}. The scale indicator is ignored if not specified in the request.
.if *! c anything*		-	If condition *c* false, accept *anything.*
.if *N anything*		-	If expression $N > 0$, accept *anything.* The scale indicator is ignored if not specified in the request.
.if *! N anything*		-	If expression $N \leqslant 0$ accept *anything.* The scale indicator is ignored if not specified in the request.
.if *'string1 ' string2 ' anything*			If *string1* is identical to *string2*, accept *anything.*
.if *! 'string1 ' string2 ' anything*			If *string1* is not identical to *string2*, accept *anything.*

Table U

ENVIRONMENT SWITCHING REQUEST

REQUEST FORM	*INITIAL VALUE*	*IF NO ARGUMENT*	*EXPLANATION*
.ev *N*	*N* = 0	previous	Environment switched to 0, 1, or 2. Switching is done in pushdown fashion so that restoring a previous environment must be done with **.ev** rather than specific reference.

Table V

INSERTIONS FROM STANDARD INPUT REQUESTS

REQUEST FORM	*INITIAL VALUE*	*IF NO ARGUMENT*	*EXPLANATION*
.ex	-	-	Exit from the **nroff/troff** formatter. Text processing is terminated exactly as if all input had ended.
.rd *prompt*	-	prompt=BEL	Read insertion from the standard input until two newline characters in a row are found. If standard input is the user keyboard, a *prompt* (or a BEL) is written onto the user terminal. The request behaves like a macro; arguments may be placed after *prompt*.

Table W

INPUT/OUTPUT FILE SWITCHING REQUESTS

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*.
.pi *program*	-	Pipe output to *program* (**nroff** formatter only). This request must occur before any printing occurs. No arguments are transmitted to *program*.
.so *filename*	-	Switch source file (pushdown). The top input level (file reading) is switched to *filename*. Contents are interpolated at the point the request is encountered. When the new file ends, input is again taken from the original file. The **.so** requests may be nested.

Table X

MISCELLANEOUS REQUESTS

REQUEST FORM	*INITIAL VALUE*	*IF NO ARGUMENT*	*EXPLANATION*
.co	-	-	Specify the point in the macro file at which compaction ends. When **-k***name* is called on the command line, all lines in the file *name* before the **.co** request will be compacted.
.fl	-	-	Flush output buffer. Used in interactive debugging to force output. The request causes a break.
.ig *yy*	-	*.yy* = ..	Ignore input lines until call of *yy*. This request behaves like the **.de** request except that the input is discarded. The input is read in *copy* mode, and any automatically incremented registers will be affected.
.mc *c N*	-	off	Sets margin character *c* and separation *N*. Specifies that a margin character *c* appear a distance *N* to the right of the right margin after each nonempty text line (except those produced by **.tl**). If the output line is too long (as can happen in no-fill mode), the character will be appended to the line. If *N* is not given, the previous *N* is used; the initial *N* is 0.2 inches in the **nroff** formatter and 1 em in **troff**. Relevant parameters are a part of the current environment. The scale indicator is ignored if not specified in the request.

MISCELLANEOUS REQUESTS

REQUEST FORM	*INITIAL VALUE*	*IF NO ARGUMENT*	*EXPLANATION*
.pm *t*	-	all	Print macros. The names and sizes of all defined macros and strings are printed on the user terminal. If *t* is given, only the total of the sizes is printed. Sizes are given in blocks of 128 characters.
.tm *string*		newline	Print *string* on terminal (UNIX operating system standard message output). After skipping initial blanks, string (rest of the line) is read in *copy* mode and written on the user terminal.

Table Y

OUTPUT AND ERROR MESSAGES REQUEST

REQUEST FORM	*INITIAL VALUE*	*IF NO ARGUMENT*	*EXPLANATION*
.ab *text*	-	-	Prints *text* on the message output and terminates without further processing. If *text* is missing, "User Abort." is printed. This request does not cause a break. The output buffer is flushed.

Table Z

NAMING CONVENTION FOR THE MATHEMATICS TYPESETTING PROGRAM

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

NAMING CONVENTION FOR THE MATHEMATICS TYPESETTING PROGRAM

CHARACTER SEQUENCE	*OUTPUT*
DELTA	Δ
GAMMA	Γ
LAMBDA	Λ
OMEGA	Ω
PHI	Φ
PI	Π
PSI	Ψ
SIGMA	Σ
THETA	Θ
UPSILON	Υ
XI	Ξ
alpha	α
beta	β
chi	χ
delta	δ
epsilon	ϵ
eta	η
gamma	γ
iota	ι
kappa	κ
lambda	λ
mu	μ
nu	ν
omega	ω
omicron	o
phi	ϕ
pi	π
psi	ψ
rho	ρ

NAMING CONVENTION FOR THE MATHEMATICS TYPESETTING PROGRAM

CHARACTER SEQUENCE	*OUTPUT*
sigma	σ
tau	τ
theta	θ
upsilon	υ
xi	ξ
zeta	ζ

MEMORANDUM MACROS

INTRODUCTION

Purpose

This section 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** (refer to **troff**(1) in the *UNIX Programmer's Manual—Volume 1: Commands and Utilities* for more details).

Conventions

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

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.

Examples may show both **nroff** and **troff** formatter outputs (of files using MM macros). In those cases in which the behavior of the two formatters 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.

Document Structure

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

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

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

• *Body segment* is the actual text of the document. It may be as small as a single paragraph or as large as hundreds of pages. It may have a hierarchy of headings up to seven levels deep. 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. The body may also contain various types of displays, tables, figures, references, and footnotes.

• *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). 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.

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.

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.

Definitions

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

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

```
.sp
```

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

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

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

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

may result in the following output:

```
Today is December 12, 1984.
```

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

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

by invoking a macro designed for that purpose. Table BB is an alphabetical list of string names used by MM. A brief description, paragraph reference, and initial (default) value(s) are given for each.

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

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

Table CC is an alphabetical list of number register names giving for each a brief description, paragraph reference, initial (default) value, and legal range of values (where [m:n] means values from m to n, inclusive).

Tables AA, BB, and CC list all macros, strings, and number registers defined in MM.

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.

The mm Command

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

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

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

OPTION	*MEANING*
−e	The **neqn** is to be invoked; also causes **neqn** to read */usr/pub/eqnchar* [see **eqnchar**(7)].
−t	The **tbl**(1) processor is to be invoked.
−c	The **col**(1) postprocessor is to be invoked.
−E	The −e option of the **nroff** formatter is to be invoked.
−y	The −mm (uncompacted macros) is to be used instead of −cm.
−12	The 12-pitch mode is to be used. The pitch switch on the terminal should be set to 12 if necessary.
−T450	Output is to a DASI 450. This is the default terminal type [unless $TERM is set; see **sh**(1)]. It is also equivalent to −T1620.
−T450−12	Output is to a DASI 450 in 12-pitch mode.
−T300	Output is to a DASI 300 terminal.
−T300−12	Output is to a DASI 300 in 12-pitch mode.
−T300s	Output is to a DASI 300S.
−T300s−12	Output is to a DASI 300S in 12-pitch mode.

−T4014	Output is to a Tektronix 4014.
−T37	Output is to a *TELETYPE* Model 37.
−T382	Output is to a DTC-382.
−T4000a	Output is to a Trendata 4000A.
−TX	Output is prepared for an EBCDIC line printer.
−Thp	Output is to a HP264x (implies −c).
−T43	Output is to a *TELETYPE*® Model 43 (implies −c).
−T40/4	Output is to a *TELETYPE* Model 40/4 (implies −c).
−T745	Output is to a Texas Instrument 700 series terminal (implies −c).
−T2631	Output is prepared for a HP2631 printer where −T2631−e and −T2631−c may be used for expanded and compressed modes, respectively (implies −c).
−Tlp	Output is to a device with no reverse or partial line motions or other special features (implies −c).

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

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

The −cm or −mm Flag

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

- defines the Memorandum Macros

- sets default values for various parameters

• initializes the formatter to be ready to process input text files.

Typical Command Lines

The prototype command lines are as follows:

• Text without tables or equations:

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

mmt [options] file ...
or
troff [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] −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] −cm
```

• Text with both tables and equations:

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

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

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

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

```
mm 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 ...
```

or 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 HP264x CRT family:

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

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

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

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

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

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. *n* = 2 permits use of Bell logo, if available, on a printing device (currently available for Xerox 9700 only).

−rC*n*	sets type of copy (e.g., DRAFT) to be printed at bottom of each page. *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. *n* = 0 (default), all pages get the prevailing header. *n* = 1, page header replaces footer on page 1 only. *n* = 2, page header is omitted from page 1. *n* = 3, "section-page" numbering occurs (.FD 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 2ff*
0	header	header
1	header replaces footer	header
2	no header	header
3	"section-page" as footer	same as page 1
4	no header	no header unless .PH defined
5	"section-page" as footer and "section-figure"	same as page 1

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

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

Omission of −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
.so /usr/lib/tmac/tmac.m
remainder of text
```

In this case, the user must not use the −cm or −mm flag [nor the **mm**(1) or **mmt**(1) command]; the **.so** request has the equivalent effect, but registers shown in 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.

FORMATTING CONCEPTS

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 as necessary. It is possible to turn any of these modes on and off (.SA, *Hy*, 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 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.

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). 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.

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.

Hyphenation

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

```
.nr Hy 1
```

once at the beginning of the document input file. 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
```

Tabs

Macros .MT, .TC, and .CS 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.

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

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, headings, and lists. Users who include BEL characters in their input text file (especially in arguments to macros) will receive mangled output.

Bullets

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

```
\*(BU
```

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

Dashes, Minus Signs, and Hyphens

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

• Users who intend to use the **nroff** formatter only may use the minus sign (–) for the minus, hyphen, and dash.

• Users who plan to use the **troff** formatter primarily should follow **troff** escape conventions.

• 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 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.

Trademark String

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

```
The
.I
UNIX
.R
\*(Tm
.I
Programmer's Manual
.R
is available from the library.
```

yields:

The *UNIX* ™ *Programmer's Manual* is available from the library.

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.

PARAGRAPHS AND HEADINGS

Paragraphs

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

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

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. Default paragraph type is controlled by the *Pt* number register.

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

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

```
.nr Pt 1
```

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

```
.nr Pt 2
```

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

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

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

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

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.

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).

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 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.

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 a bold 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 a bold 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 ($3 \leqslant n \leqslant 7$). The following text begins on the same line, i.e., these are *run-in* headings.

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

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”.

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.

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.

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 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 run into the following text.

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

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

• If *Hi* is 0, text is left-justified.

• If *Hi* is 1 (the default value), text is indented according to the paragraph type as specified by the *Pt* register.

• 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
```

Centered Headings

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

Bold, Italic, and Underlined Headings

Control by Level:

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

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

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

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

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.

Heading Point Sizes:

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

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

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

```
.ds HP 12 12 10 10 10 10 10
```

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

```
.ds HP +2 +2 −1 −1
```

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

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

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

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
```

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.

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 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 and footnotes. If this happens, the "Out of temp file space" formatter error message will be issued; the only remedy is to save fewer levels and/or to have fewer words in the heading text.

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. 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.

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, *dlevel* is equal to the contents of register *Hu*, and *rlevel* is 0. In both cases, *heading-text* is the text of the original invocation.

By the time .H calls .HX, it has already incremented the heading counter of the specified level, produced blank lines (vertical spaces) to precede the heading, 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. 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.

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"
```

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
```

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

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.

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.

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

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

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

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

List-Initialization Macros

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

.AL	Automatically Numbered or Alphabetized List
.BL	Bullet List
.DL	Dash List
.ML	Marked List
.RL	Reference List
.VL	Variable-Item List

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.

Bullet List

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

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

• If the *text-indent* argument is non-null, it overrides the default indentation (the amount of paragraph indentation as given in the *Pi* register). 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.

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). 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.

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.

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.

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".
.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”.
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.

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.

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.

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).

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 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.

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. This is summarized in the following table:

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

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

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

where *x* is the generated number or letter.

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

• The *LI-space* argument gives the number of blank lines (halves of a vertical space) that should be output by each .LI macro in the list. If omitted, *LI-*

space defaults to 1; the value 0 can be used to obtain compact lists. If *LI-space* is greater than 0, the .LI macro issues a **.ne** request for two lines just before printing the mark.

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

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

• The normal case is to set *LI-space* to 1 and *LB-space* to 0 yielding one blank line before each item in the list; such a list is usually terminated with a .LE 1 macro to end the list with a blank line.

• For a more compact list, *LI-space* is set to 0, *LB-space* is set to 1, and the .LE 1 macro is used at the end of the list. The result is a list with one blank line before and after it.

• If both *LI-space* and *LB-space* are set to 0 and the .LE macro is used to end the list, a list without any blank lines will result.

User-Defined List Structures

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

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

```
A.

    [1]

        •

            a)

                +
```

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

```
.\"register g is used as a local temporary to save :g before it is changed
                                  below
.de aL
.nr g \\n(:g
.if \\ng=0 .AL A                  \"produces an A.
.if \\ng=1 .LB \\n(Li 0 1 4       \"produces a [1]
.if \\ng=2 .BL                    \"produces a bullet
.if \\ng=3 .LB \\n(Li 0 2 2 a     \"produces an a)
.if \\ng=4 .ML +                  \"produces a +
..
```

This macro can be used (in conjunction with .LI and .LE) instead of .AL, .RL, .BL, .LB, and .ML. For example, the following input:

```
.aL
.LI
First line.
.aL
.LI
Second line.
.LE
.LI
Third line.
.LE
```

when formatted yields

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

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

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

when formatted yields

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

    A.  First line.

        [1] Second line.

    B.  Third line.

    C.  Fourth line.
    Fifth line.
```

MEMORANDUM AND RELEASED-PAPER STYLE DOCUMENTS

One use of MM is for the preparation of memoranda and released-paper documents (a documentation style used by AT&T 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.

Sequence of Beginning Macros

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

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

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

If neither the memorandum nor released-paper style is desired, the TL, AU, TM, AE, OK, MT, ND, and AF macros should be omitted from the input text. If these macros are omitted, the first page will have only the page header followed by the body of the document.

Title

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

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

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

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

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

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

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

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

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

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 on a separate line following the signer's name. The .AT must immediately follow the .AU for the given author. For example:

```
.AU "S. V. Earhart" SVE SF 4541 6037 5-219
.AT Planner "Documentation Services"
```

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 "W. J. Salica" WJS SF 4541 6380 5-219 4541-543210.01MF
```

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, "AT&T Bell Laboratories" and the location are centered. The paragraph on memorandum types contains information regarding authors from different locations.

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.

Abstract

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

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

- released paper
- technical memorandum
- memorandum for file (also used for engineer's notes, memoranda for record, etc.).

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

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

0 — Abstract will be printed on page 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 are allowed on memorandum for file cover sheets; the .NS and .NE macros must appear after the .AS 2 and .AE macros. Headings and displays are not permitted within an abstract.

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*.

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.

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
i	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 "Steven Earhart"
```

produces

```
    Steven Earhart - 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 "AT&T 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 location codes are defined as strings that are the addresses of the corresponding 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 and their associated lines of input are ignored when the released-paper style is specified.

Authors from locations may include their affiliations in the released-paper style by specifying the appropriate .AF macro 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 "S. Earhart" "" ZZ
.AF "AT&T 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.

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.

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

The .AF with no argument suppresses "AT&T 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, 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 are not effective with .AF. The only .AF use appropriate for the **troff** formatter is to specify a replacement for "AT&T Bell Laboratories".

The command line option −rE*n* controls the font of the "Subject/Date/From" block.

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
```

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.

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
```

"Copy to" and Other Notations

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

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

Codes for *arg* and the corresponding notations are:

arg	*NOTATIONS*
none	Copy to
""	Copy to
0	Copy to
1	Copy (with att.) to
2	Copy (without att.) to
3	Att.
4	Atts.
5	Enc.
6	Encs.
7	Under Separate Cover
8	Letter to
9	Memorandum to
"*string*"	Copy (string) to

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

```
.NS "with att. 1 only"
```

will generate

```
Copy (with att. 1 only) to
```

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

```
.NS 4
Attachment 1-List of register names
Attachment 2-List of string and macro names
.NS 1
S. V. Earhart
.NS 2
W. J. Salica
G. H. Hurtz
.NE
```

would be formatted as

```
Atts.
Attachment 1-List of register names
Attachment 2-List of string and macro names

Copy (with att.) to
S. V. Earhart

Copy (without att.) to
W. J. Salica
G. H. Hurtz
```

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

Approval Signature Line

```
.AV approver's-name
```

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

```
     .AV "Jane Doe"
```

produces

APPROVED:

____ ____________________

Jane Doe   Date

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.

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 may never be nested in any combination. Although lists and paragraphs are permitted, no headings (.H or .HU) can occur within displays or footnotes.

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. 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. 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."

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 may also be used to control handling of displays in double-column mode and to control the break in text before floating displays.

Tables

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

The **.TS** (table start) and **.TE** (table end) macros make possible the use of the **tbl**(1) program. These macros are used to delimit text to be examined by **tbl** and to set proper spacing around the table. The display function and the **tbl** delimiting function are independent. In order to permit the user to keep together blocks that contain any mixture of tables, equations, filled text, unfilled text, and caption lines, the .TS/.TE block should be enclosed within a display (.DS/.DE). Floating tables may be enclosed inside floating displays (.DF/.DE).

Macros .TS and .TE permit processing of tables that extend over several pages. If a table heading is needed for each page of a multipage table, the "H" argument should be specified to the .TS macro as above. Following the options and format information, table title is typed on as many lines as required and is followed by the .TH macro. The .TH macro must occur when ".TS H" is used for a multipage table. This is not a feature of **tbl** but of the definitions provided by the 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**(1)] may be used for terminals, such as the 450, that are capable of finer printing resolution. This will cause better alignment of features such as the lines forming the corner of a box. The −e is not effective with *col*(1).

Equations

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

Mathematical typesetting programs **eqn**(1) and **neqn** expect to use the **.EQ** (equation start) and **.EN** (equation end) macros as delimiters in the same way that **tbl**(1) uses .TS and .TE; however, .EQ and .EN must occur inside a .DS/.DE pair. There is an exception to this rule — if .EQ and .EN are used to specify only the delimiters for in-line equations or to specify **eqn/neqn** defines, the .DS and .DE macros must not be used; otherwise, extra blank lines will appear in the output.

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

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

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 −rN5 to reset counters in sections). 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* argumnet is omitted or 0, *override* is used as a prefix to the number; if the *flag* argument is 1, *override* is used as a suffix; and if the *flag* argument is 2, *override* replaces the number. If −rN5 is given, "section-figure" numbering is set automatically and 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.

List of Figures, Tables, Equations, and Exhibits

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

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

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

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.

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.

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 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 to "*F" or *label* when they follow an end-of-sentence punctuation mark (i.e., period, question mark, exclamation point).

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.

			TEXT	*LABEL*
arg	*HYPHENATION*	*ADJUST*	*INDENT*	*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.

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.

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*.

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 previously.

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.

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.

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, 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) followed by the current line number within the current input file.

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.

Odd-Page Header

```
.OH [arg]
```

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

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, the type of copy follows the footer on a separate line. In particular, if −rC3 or −rC4 (DRAFT) is specified, the footer is initialized to contain the date instead of being a blank line.

Even-Page 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.

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.

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.

Default Header and Footer With Section-Page Numbering

Pages can be numbered sequentially within sections by "section-number page-number". 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.

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:

```
.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.

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'"
```

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. Formatting done within the .TP macro is processed in an environment different from that of the body. For example, to obtain a page header that includes three centered lines of data, i.e., document number, issue date, and revision date, the user could define the .TP as follows:

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

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

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.

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).

Proprietary Marking

.PM [code]

The **.PM** (proprietary marking) macro appends to the page footer a PRIVATE, NOTICE, AT&T BELL LABORATORIES PROPRIETARY, or AT&T BELL LABORATORIES RESTRICTED disclaimer. The *code* argument may be:

code	***DISCLAIMER***
none	turn off previous disclaimer, if any
P	PRIVATE
N	NOTICE
BP	AT&T BELL LABORATORIES PROPRIETARY
BR	AT&T BELL LABORATORIES RESTRICTED

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

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.

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.

These macros normally appear once at the end of the document, after the Signature Block and Notations 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.

Table of Contents

```
.TC [slevel] [spacing] [tlevel] [tab] [head1] [head2] [head3] [head4] [head5]
```

The **.TC** macro generates a table of contents containing heading levels that were saved for the table of contents as determined by the value of the *Cl* register. 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 (*head1* ... *head5*) are horizontally centered on the page and precede the table of contents.

If the .TC macro is invoked with at most four arguments, the user-exit macro .TX is invoked (without arguments) before the word "CONTENTS" is printed or the user-exit macro .TY is invoked and the word "CONTENTS" is not printed.

By defining .TX or .TY and invoking .TC with at most four arguments, the user can specify what needs to be done at the top of the first page of the table of contents. For example:

```
.de TX
.ce 2
Special Application
Message Transmission
.sp 2
.in +10n
Approved: \l'3i'
.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.

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. All other information for the cover sheet is obtained from data given before the .MT macro call. 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.

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.

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.

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
```

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.

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.

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

arg	*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"
```

MISCELLANEOUS FEATURES

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**(1) 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 " text " right -justified
```

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.

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.

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 on the command line. This causes the *RE* string to be output as part of the page header.

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.

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**.

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 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
leftcenterrightleftcenterright
first columnsecond 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.

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" immediately followed by ".P" 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.

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.

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.

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*) 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 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

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

Producing Accents

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

	INPUT	*OUTPUT*
Grave accent	c*§	c´
Acute accent	e*′	e´
Circumflex	o*^	o^
Tilde	n*	n
Cedilla	c*,	c,
Lower-case umlaut	u*:	u¨
Upper-case umlaut	U*;	U¨

Inserting Text Interactively

```
.RD [prompt] [diversion] [string]
```

The **.RD** (read insertion) macro allows a user to stop the standard output of a document and to read text from the standard input until two consecutive newline characters are found. When newline characters are encountered, normal output is resumed.

• The *prompt* argument will be printed at the terminal. If not given, .RD signals the user with a BEL on terminal output.

• The *diversion* argument allows the user to save all text typed in after the prompt in a macro whose name is that of the diversion.

• The *string* argument allows the user to save for later reference the first line following the prompt in the named string.

The .RD macro follows the formatting conventions in effect. Thus, the following examples assume that the .RD is invoked in no fill mode (**.nf**):

```
.RD Name aA bB
```

produces

```
Name: S. Earhart   (user types name)
16 Elm Rd.,
Piscataway
```

The diverted macro .aA will contain

```
S. Earhart
16 Elm Rd.,
Piscataway
```

The string *bB* (*(*bB*) contains “S. Earhart”.

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

ERRORS AND DEBUGGING

Error Terminations

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

• A break occurs.

• The formatter output buffer (which may contain some text) is printed to avoid confusion regarding location of the error.

• A short message is printed giving the name of the macro that detected the error, type of error, and approximate line number in the current input file of the last processed input line. Error messages are explained in Table 4.D.

• Processing terminates unless register D has a positive value. In the latter case, processing continues even though the output is guaranteed to be deranged from that point on.

The error message is printed by outputting the message directly to the user terminal. If an output filter, such as **300**(1), **450**(1), or **hp**(1) is being used to post-process the **nroff** formatter output, the message may be garbled by being intermixed with text held in that filter's output buffer.

Note: If any of **cw**(1), **eqn**(1)/**neqn**, and **tbl**(1) programs are being used and if the **-o***list* option of the formatter causes the last page of the document not to be printed, a harmless "broken pipe" message may result.

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.

EXTENDING AND MODIFYING MM MACROS

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.

Names Used by Formatters

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

Names Used by 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)
	)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)

Names Used by cw, eqn/neqn, and tbl

The *cw*(1) program is the constant-width font preprocessor for the **troff** formatter. It uses the following five macro names:

.CD .CN .CP .CW .PC

This preprocessor also uses the number register names *cE* and *cW*. Mathematical equation preprocessors, **eqn**(1) and **neqn**, use registers and string names of the form *nn*. The table preprocessor, **tbl**(1), uses T&, T#, and TW, and names of the form:

a− a+ a| nn na ^a #a #s

Names Defined by User

Names that consist 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)

Sample Extensions

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 apppendix titles the *Hc* register must be set to 1.

Hanging Indent With Tabs.

The following example illustrates the use of the hanging indent feature of variable-item lists. 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.
```

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**(1) formatter so that output can be produced on both a phototypesetter and a teletypewriter/CRT terminal.

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

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

Table AA

MM MACRO NAMES SUMMARY

MACRO	*DESCRIPTION*
1C	1-column processing .1C
2C	2-column processing .2C
AE	Abstract end .AE
AF	Alternate format of "Subject/Date/From" block .AF [company-name]
AL	Automatically incremented list start .AL [type] [text-indent] [1]
AS	Abstract start .AS [arg] [indent]
AT	Author's title .AT [title] ...
AU	Author information .AU name [initials] [loc] [dept] [ext] [room] [arg] [arg] [arg]
AV	Approval signature .AV [name]
B	Bold .B [bold-arg] [previous-font-arg] [bold] [prev] [bold] [prev]
BE	Bottom block end .BE
BI	Bold/Italic .BI [bold-arg] [italic-arg] [bold] [italic] [bold] [italic]
BL	Bullet list start .BL [text-indent] [1]

MM MACRO NAMES SUMMARY

MACRO	*DESCRIPTION*
BR	Bold/Roman .BR [bold-arg] [Roman-arg] [bold] [Roman] [bold] [Roman]
BS	Bottom block start .BS
CS	Cover sheet .CS [pages] [other] [total] [figs] [tbls] [refs]
DE	Display end .DE
DF	Display floating start .DF [format] [fill] [right-indent]
DL	Dash list start .DL [text-indent] [1]
DS	Display static start .DS [format] [fill] [right-indent]
EC	Equation caption .EC [title] [override] [flag]
EF	Even-page footer .EF [arg]
EH	Even-page header .EH [arg]
EN	End equation display .EN
EQ	Equation display start .EQ [label]
EX	Exhibit caption .EX [title] [override] [flag]

MM MACRO NAMES SUMMARY

MACRO	*DESCRIPTION*
FC	Formal closing .FC [closing]
FD	Footnote default format .FD [arg] [1]
FE	Footnote end .FE
FG	Figure title .FG [title] [override] [flag]
FS	Footnote start .FS [label]
H	Heading—numbered .H level [heading-text] [heading-suffix]
HC	Hyphenation character .HC [hyphenation-indicator]
HM	Heading mark style (Arabic or Roman numerals, or letters) .HM [arg1] ... [arg7]
HU	Heading—unnumbered .HU heading-text
HX*	Heading user exit X (before printing heading) .HX dlevel rlevel heading-text
HY*	Heading user exit Y (before printing heading) .HY dlevel rlevel heading-text

* 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.

MM MACRO NAMES SUMMARY

MACRO	*DESCRIPTION*
HZ*	Heading user exit Z (after printing heading) .HZ dlevel rlevel heading-text
I	Italic (underline in the **nroff** formatter) .I [italic-arg] [previous-font-arg] [italic] [prev] [italic] [prev]
IB	Italic/Bold .IB [italic-arg] [bold-arg] [italic] [bold] [italic] [bold]
IR	Italic/Roman .IR [italic-arg] [Roman-arg] [italic] [Roman] [italic] [Roman]
LB	List begin .LB text-indent mark-indent pad type [mark] [LI-space] [LB-space]
LC	List-status clear .LC [list-level]
LE	List end .LE [1]
LI	List item .LI [mark] [1]
ML	Marked list start .ML mark [text-indent] [1]
MT	Memorandum type .MT [type] [addressee] *or* .MT [4] [1]
ND	New date .ND new-date
NE	Notation end .NE
NS	Notation start .NS [arg]

* 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.

MM MACRO NAMES SUMMARY

MACRO	*DESCRIPTION*
nP	Double-line indented paragraphs .nP
OF	Odd-page footer .OF [arg]
OH	Odd-page header .OH [arg]
OK	Other keywords for the Technical Memorandum cover sheet .OK [keyword] ...
OP	Odd page .OP
P	Paragraph .P [type]
PF	Page footer .PF [arg]
PH	Page header .PH [arg]
PM	Proprietary Marking .PM [code]
PX*	Page-header user exit .PX
R	Return to regular (Roman) font .R
RB	Roman/Bold .RB [Roman-arg] [bold-arg] [Roman] [bold] [Roman] [bold]

* 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.

MM MACRO NAMES SUMMARY

MACRO	*DESCRIPTION*
RD	Read insertion from terminal .RD [prompt] [diversion] [string]
RF	Reference end .RF
RI	Roman/Italic .RI [Roman-arg] [italic-arg] [Roman] [italic] [Roman] [italic]
RL	Reference list start .RL [text-indent] [1]
RP	Produce Reference Page .RP [arg] [arg]
RS	Reference start .RS [string-name]
S	Set **troff** formatter point size and vertical spacing .S [size] [spacing]
SA	Set adjustment (right-margin justification) default .SA [arg]
SG	Signature line .SG [arg] [1]
SK	Skip pages .SK [pages]
SM	Make a string smaller .SM string1 [string2] [string3]
SP	Space vertically .SP [lines]

* 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.

MM MACRO NAMES SUMMARY

MACRO	*DESCRIPTION*
TB	Table title .TB [title] [override] [flag]
TC	Table of contents .TC [slevel] [spacing] [tlevel] [tab] [head1] [head2] [head3] [head4] [head5]
TE	Table end .TE
TH	Table header .TH [N]
TL	Title of memorandum .TL [charging-case] [filing-case]
TM	Technical Memorandum number(s) .TM [number] ...
TP*	Top-of-page macro .TP
TS	Table start .TS [H]
TX*	Table of contents user exit .TX
TY*	Table of contents user exit (suppresses "CONTENTS") .TY
VL	Variable-item list start .VL text-indent [mark-indent] [1]
VM	Vertical margins .VM [top] [bottom]
WC	Footnote and Display Width Control .WC [format]

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

Table BB

STRING NAMES SUMMARY

STRING	*DESCRIPTION*
BU	Bullet NROFF: ⊕ TROFF: ●
Ci	Table of contents indent list Up to seven scaled arguments for heading levels
DT	Date Current date, unless overridden Month, day, year (e.g., May 31, 1979)
EM	Em dash string Produces an em dash in the **troff** formatter and a double hyphen in **nroff**
F	Footnote number generator NROFF: \u\\n+(:p\d TROFF: \v'−.4m'\s−3\\n+(:p\s0\v'.4m'
HF	Heading font list Up to seven codes for heading levels 1 through 7 3 3 2 2 2 2 2 (levels 1 and 2 bold, 3 through 7 underlined by **nroff** and italicized by **troff**)
HP	Heading point size list Up to seven codes for heading levels 1 through 7
Le	Title for LIST OF EQUATIONS
Lf	Title for LIST OF FIGURES
Lt	Title for LIST OF TABLES
Lx	Title for LIST OF EXHIBITS
RE	SCCS Release and Level of MM Release.Level (e.g., 15.129)

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

STRING NAMES SUMMARY

STRING	*DESCRIPTION*
Rf	Reference numberer
Rp	Title for references
Tm	Trademark string Places the letters "TM" one-half line above the text that it follows
	Seven accent strings are also available

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

Table CC

NUMBER REGISTER NAMES SUMMARY

REGISTER	*DESCRIPTION*
A *†	Handles preprinted forms and AT&T logo 0, [0:2]
Au	Inhibits printing of author information 1, [0:1]
C *†	Copy type (original, DRAFT, etc.) 0 (Original), [0:4]
Cl	Level of headings saved for table of contents 2, [0:7]
Cp	Placement of list of figures, etc. 1 (on separate pages), [0:1]
D *†	Debug flag 0, [0:1]
De	Display eject register for floating dislays 0, [0:1]
Df	Display format register for floating displays 5, [0:5]
Ds	Static display pre- and post-space 1, [0:1]
E *†	Controls font of the Subject/Date/From fields 1 **(nroff)** 0 **(troff)**, [0:1]

* 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.

† Any register having a single-character name can be set from the command line.

NUMBER REGISTER NAMES SUMMARY

REGISTER	*DESCRIPTION*
Ec	Equation counter, used by .EC macro 0, [0:?], incremented by one for each .EC call.
Ej	Page-ejection flag for headings 0 (no eject), [0:7]
Eq	Equation label placement 0 (right-adjusted), [0:1]
Ex	Exhibit counter, used by .EX macro 0, [0:?], incremented by one for each .EX call.
Fg	Figure counter, used by .FG macro 0, [0:?], incremented by one for each .FG call.
Fs	Footnote space (i.e., spacing between footnotes) 1, [0:?]
H1-H7	Heading counters for levels 1 through 7 0, [0:?], incremented by the .H macro of corresponding level or the .HU macro if at level given by the **Hu** register. The H2 through H7 registers are reset to 0 by any .H (.HU) macro at a lower-numbered level.
Hb	Heading break level (after .H and .HU) 2, [0:7]
Hc	Heading centering level for .H and .HU 0 (no centered headings), [0:7]
Hi	Heading temporary indent (after .H and .HU) 1 (indent as paragraph), [0:2]
Hs	Heading space level (after .H and .HU) 2 (SPACE ONLY AFTER .H 1 AND .H 2), [0:7]

* 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.

† Any register having a single-character name can be set from the command line.

NUMBER REGISTER NAMES SUMMARY

REGISTER	*DESCRIPTION*
Ht	Heading type (for .H: single or concatenated numbers) 0 (concatenated numbers: 1.1.1, etc.), [0:1]
Hu	Heading level for unnumbered heading (.HU) 2 (.HU AT THE SAME LEVEL AS.H 2), [0:7]
Hy	Hyphenation control for body of document 0 (automatic hyphenation off), [0:1]
L *†	Length of page 66, [20:?] (11i, [2i:?] in **troff** formatter)
Le	List of equations 0 (list not produced) [0:1]
Lf	List of figures 1 (list produced) [0:1]
Li	List indent 6 (**nroff**) 5 (**troff**), [0:?]
Ls	List spacing between items by level 6 (spacing between all levels) [0:6]
Lt	List of tables 1 (list produced) [0:1]
Lx	List of exhibits 1 (list produced) [0:1]
N *†	Numbering style 0, [0:5]

* 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.

† Any register having a single-character name can be set from the command line.

NUMBER REGISTER NAMES SUMMARY

REGISTER	*DESCRIPTION*
Np	Numbering style for paragraphs 0 (unnumbered) [0:1]
O *†	Offset of page .75i, [0:?] (0.5i, [0i:?] in **troff** formatter) For **nroff** formatter, these values are unscaled numbers representing lines or character positions. For **troff** formatter, these values must be scaled.
Oc	Table of contents page numbering style 0 (lowercase Roman), [0:1]
Of	Figure caption style 0 (period separator), [0:1]
P†	Page number managed by MM 0, [0:?]
Pi	Paragraph indent 5 (**nroff**) 3 (**troff**), [0:?]
Ps	Paragraph spacing 1 (one blank space between paragraphs), [0:?]
Pt	Paragraph type 0 (paragraphs always left justified), [0:2]
Pv	"PRIVATE" header 0 (not printed), [0:2]
Rf	Reference counter, used by .RS macro 0, [0:?], incremented by one for each .RS call.
S *†	The **troff** formatter default point size 10, [6:36]

* 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.

† Any register having a single-character name can be set from the command line.

NUMBER REGISTER NAMES SUMMARY

REGISTER	*DESCRIPTION*
Si	Standard indent for displays 5 (**nroff**) 3 (**troff**), [0:?]
T *†	Type of **nroff** output device 0, [0:2]
Tb	Table counter, used by .TB macro 0, [0:?], incremented by one for each .TB call.
U *†	Underlining style (**nroff**) for .H and .HU 0 (continuous underline when possible), [0:1]
W *†	Width of page (line and title length) 6i, [10:1365] (6i, [2i:7.54i] in the **troff** formatter)

* 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.

† Any register having a single-character name can be set from the command line.

VIEWGRAPHS AND SLIDES MACROS

INTRODUCTION

This section describes a package of UNIX operating system **troff**(1)[1] 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**(1), and the **troff** formatter.

With the MV macros, viewgraphs can be prepared in a variety of dimensions, as well as 35mm slides and 2x2 "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 operating 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**(1), or preprocessed by **eqn**(1), **tbl**(1), **cw**(1), etc.

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.

MACROS

The following is an explanation of the MV macros which are summarized in **mv**(1) of the *UNIX Programmer's Manual—Volume 1: Commands and Utilities*.

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 Table DD.

1. The notation **name**(*N*) indicates entry **name** in Section *N* of the *UNIX Programmer's Manual.*

VIEWGRAPHS AND SLIDES MACROS

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 7x5.4 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*
- Company
- 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.

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.

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 a half-line 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.

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-line from the preceding text.

The .B level *mark* may be changed by specifying the desired character string[2] as the first argument. 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.

2. All character-string arguments that contain spaces must be quoted ("...").

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.

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 following text to start on a new line. In other words, it causes a break. 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.

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 prevailing point size. Any indentation established by the .I macro has no effect on titles; they are always centered within the foil horizontal dimension.

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 the *NROFF and TROFF User's Manual* section 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 before exiting. The third argument, if present, is passed to the .A macro.

Point Sizes and Line Lengths

.S [ps] [ll]

Each foil-start macro begins the foil with an appropriate default point size[3] and line length. 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 (Table EE). 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 less than 10, it is taken as inches. If it is not dimensioned and greater than or equal to 10, it is taken as **troff** formatter units (1/432nds of an inch).

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

3. Default point sizes for each type of foil and corresponding maximum number of lines are given in Table EE.

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
```

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. 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 non-null arguments must be dimensioned. Null arguments leave the corresponding spacing unaffected. The initial setting is equivalent to

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

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

<u>phototypesetter</u>

while

.U under line

produces

<u>under</u>line

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's Manual* contains definitions of these requests.

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 .AD, .BR, .CE, .FI, .NA, .NF, .SP, and .TI also cause a break.

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.

THE TROFF PREPROCESSORS

It is possible to use the various **troff** formatter preprocessors to typeset foils that require more powerful formatting capabilities.

Tables

The **tbl**(1) 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**. The *Table Formatting Program* **(tbl)** describes the macros used for generating tables.

Mathematical Expressions

The **eqn**(1) program can be used to typeset mathematical expressions and formulas on foils provided care is taken to specify proper fonts and point sizes. The *Mathematics Typesetting Program* (**eqn**) describes the macros used for processing equations. The .EQ and .EN macros are not defined in the MV macro package.

Constant-Width Program Examples

The constant-width font simulates computer-terminal and line-printer output and can be sometimes effective in presenting computer-related topics. The **cw**(1) program illustrates the preprocessor.

FINISHED PRODUCT

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)
−e	invoke **eqn**(1)
−t	invoke **tbl**(1)
−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 **cw**(1) preprocessor:

```
cw [ options ] file ... | mvt [ options ] -
```

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, text will be printed after the "cross hairs."

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 very simple processes (e.g., 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.

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 (Table EE) 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:[4]

```
.DF 1 H 2 HI 3 HM
```

or the Times Roman regular, italic, and bold:

```
.DF 1 R 2 I 3 B
```

should be mounted via the .DF macro. Bold typefaces tend to be a bit overwhelming. Choice of fonts is primarily a matter of personal aesthetics.

• 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 (i.e., increase the top margin).

• Normal uppercase and lowercase text is more legible than uppercase text only.[5] Uppercase and lowercase alphabets have evolved and 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.

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.

4. Helvetica medium is really a bold typeface.

5. 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.

- Do not underline.
- Use many "sparse" foils rather than a few "dense" ones.
- Use fewer words rather than more.
- Use larger point sizes rather than smaller.
- Use larger top and bottom margins rather than smaller.
- Use normal uppercase and lowercase text rather than uppercase text only.

WARNINGS

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. Other **troff** formatter requests should be used sparingly (if at all) and with care and discipline. Particularly dangerous are requests that affect point size, indentation, page offset, line and title lengths, and vertical spacing between lines. The .S and .I macros should be used instead.

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 are used, their reserved names must also be avoided.

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 "non-paddable" 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 ~~
```

DIMENSIONAL DETAILS

For each style of viewgraph Table EE 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.

Table DD

FOIL-START MACROS

MACRO NAME	*SIZE* AND TYPE*	*FRAME NUMBER†*
.VS	7X7 viewgraph or 2X2 super-slide	E-7351 or E-7351-R
.Vw	7X5 viewgraph	E-7351-B
.Vh	5X7 viewgraph	E-7351-A
.VW	9X7 viewgraph	E-8814 or E-9148
.VH	7X9 viewgraph	E-8814 or E-9148
.Sw	7X5 35mm slide	E-7351-B
.Sh	5X7 35mm slide	E-7351-A
.SW	9X7 35mm slide	E-8814 or E-9148
.SH	7X9 35mm slide	E-8814 or E-9148

* Size of mounting frame opening (width and height) in inches.

† stock item number.

Table EE

DEFAULT POINT SIZE, DIMENSIONS, AND ASPECT RATIOS

MACRO (NOTE 1)	*POINT SIZE*	*MAXIMUM LINES (NOTE 2)*	*NOMINAL*				*ACTUAL (TEXT)*			
			W	*H* — *(NOTE 3)* —	*AR*	$\frac{1}{AR}$	*W*	*H* — *(NOTE 3)* —	*AR*	$\frac{1}{AR}$
.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).

Select Code 320-034
ISBN 0-03-011207-9