IBM Advanced Interactive Executive for the Personal System/2 (AIX PS/2) Text Formatting Guide Version 1.1

Document Number SC23-2044-01

IBM Advanced Interactive Executive
for the Personal System/2
(AIX PS/2)

Text Formatting Guide

Version 1.1

Document Number SC23-2044-01

Edition Notice

Second Edition (March 1991)

This edition applies to Version 1.2.1 of Text Formatting System, Program Number 5713-AFD, for use with Version 1.2.1 of IBM Advanced Interactive Executive for the Personal System/2, Program Number 5713-AEQ, and to all subsequent releases until otherwise indicated in new editions or technical newsletters. Make sure you are using the correct edition for the level of the product.

Order publications through your IBM representative or the IBM branch office serving your locality. Publications are not stocked at the address given below.

A form for reader's comments appears at the back of this publication. If the form has been removed, address your comments to:

IBM Corporation, Department 52QA MS 911
Neighborhood Road
Kingston, NY 12401
U.S.A.

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

Notices

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM's product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any of IBM's intellectual property rights or other legally protectible rights may be used instead of the IBM product, program, or service. Evaluation and verification of operation in conjunction with other products, programs, or services, except those expressly designated by IBM, are the user's responsibility.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Commercial Relations, IBM Corporation, Purchase, NY 10577.

Subtopics

Trademarks and Acknowledgments

Trademarks and Acknowledgments

The following trademarks apply to this book:

AIX is a registered trademark of International Business Machine Corporation.

IBM is a registered trademark of International Business Machine Corporation.

INed is a registered trademark of INTERACTIVE Systems Corporation

Personal System/2, PS/2, and RT are registered trademarks o International Business Machines Corporation.

Portions of the code and documentation were developed at th Electrical Engineering and Computer Sciences Department at the Berkeley Campus of the University of California under the auspices of the Regents of the University of California.

About This Book

This book describes the text formatting utilities available on the Advanced Interactive Executive Operating System for the Personal System/2 (AIX PS/2). The information in this book shows you how to use **vi**, an editing program, and how to use AIX PS/2 text formatting utilities.

Subtopics

Who Should Read This Book
What You Should Know
How to Use This Book
Related Publications

Who Should Read This Book

This book is written for anyone who wants to create, edit, and format text using the utilities available on AIX PS/2. The information in this book shows you how to:

Create, revise, and store file

Format text using memorandum macro

Format text for printer and phototypesetter outpu

Format mathematics text and equation

Format table

Format text for viewgraphs and slide presentations

What You Should Know

Before you begin, you should know how to use a PS/2 and have a general knowledge of AIX PS/2 commands, shell procedures, and general text formatting and programming concepts. In addition, some programming experience may be required if you want to interface with printing hardware. The tasks in this book require you to use one of the text editing programs available on AIX PS/2, such as **ed**, **ex**, **INed**, or **vi**.

How to Use This Book

This book is divided into three parts:

"Part 1. Using the vi Editor" describes the **vi** editor and shows you how to use this editor to create, revise, and store files.

"Part 2. Text Formatting Guide" introduces you to AIX PS/2 text formatting utilities and provides instructions for formatting text with memorandum macros and with **nroff** and **troff** formatters.

"Part 3. Text Formatting Reference" is a detailed reference to the text formatting commands and formatter requests and macros.

If you already know how to use one of the text editors available on AIX PS/2, you can skip Part 1. If you are already familiar with **vi** but simply need a quick review of **vi** commands, see Chapter 6, "A Summary of vi Commands."

You can use Part 2 as a training manual. Then, as you gain more experience using the text formatting utilities, you can continue to use Part 2 for detailed information and Part 3 for brief reminders.

This book also has four appendixes, which contain supplemental information about the text formatting utilities. Appendix D lists the error messages produced by the formatters and recommends corrective action when necessary.

Notes:

1. When a formatter is referred to, the text refers to both **troff** and **nroff** formatters. In most cases, the **nroff** and **troff** formatters use the same instruction set. When the two formatters work differently, the text tells you.

2. This book was produced using IBM typesetting equipment and formatting utilities. For that reason, the examples shown may not always look exactly like the output you get using **troff** and your phototypesetter. Printers driven by the **nroff** formatter produce output that looks substantially different from typeset output. The print appearance of **nroff** output is device dependent.

Subtopics
Highlighting and Notational Conventions

Highlighting and Notational Conventions

The following type styles and notations are used throughout this book to identify certain kinds of information:

System parts
: The names for keys, commands, files, and other parts of the system are printed in bold (for example, the **mmt** command).

New terms
: New terms when first introduced in text are printed in bold italics (for example, ***requests*** are built-in commands).

Information you are to type
: Examples and characters that must be entered exactly as shown are printed in monospace font (for example, enter: **mmt -e**).

Variable information
: The names for information that you must provide are printed in italics (for example, enter: **tbl filename**).

[ ]
: Brackets indicate that the parameters enclosed between them are optional (for example, **tbl -t -e [parm]**). You do not type the brackets when you use the command. The brackets indicate only that the enclosed information is optional; you can supply a value for **parm** or you can omit it.

...
: Ellipses indicate that a preceding parameter or group of parameters can be repeated (for example, **vi filename ...**). The ellipses indicate that you can supply more than one file name.

Related Publications

For additional information, you may want to refer to the following IBM AIX Operating System publications for the PS/2:

AIX Commands Reference, SC23-2292 (Vol. 1) and SC23-2184 (Vol. 2), lists and describes the AIX/370 and AIX PS/2 Operating System commands.

AIX Messages Reference, SC23-2294, lists messages displayed by the AIX Operating System and explains how to respond to them.

AIX Programming Tools and Interfaces, SC23-2304, describes the programming environment of the AIX Operating System and includes information about operating system tools that are used to develop, compile, and debug programs.

AIX Technical Reference, SC23-2300 (Vol. 1) and SC23-2301 (Vol. 2), describes the system calls and subroutines a programmer uses to write application programs. This book also provides information about the AIX Operating System file system, special files, miscellaneous files, and the writing of device drivers.

Using the AIX Operating System, SC23-2291, shows the beginning user how to use AIX Operating System commands to do such basic tasks as log in and out of the system, display and print files, and set and change passwords. It includes information for intermediate to advanced users about how to use communication and networking facilities and write shell procedures.

Table of Contents

Tables

1.0 Part 1. Using the vi Editor

Subtopics

1.1 Chapter 1. Introduction to vi

Subtopics
1.1.1 About This Chapter
1.1.2 Overview

1.1.1 About This Chapter

This chapter briefly describes the **vi** editor, and it describes some function keys that you will use often.

1.1.2 Overview

The **vi** ("visual") editor is a *full-screen* editor that allows you to create, edit, and store text. You can edit or modify text easily on a screen-by-screen basis since **vi** allows you to scroll through the text and process editing commands relative to the position of the cursor in the file. The line on which the cursor is located is called the ***current line***.

In addition, you can execute system functions while remaining in the editor; and you can customize the characteristics of an editing session by changing the editor's defaults.

The **vi** editor operates in two modes: command mode and input mode. In command mode, you can issue commands to perform a number of editing tasks including: inserting text, making corrections or deleting text, undoing the changes made, and storing the file you are editing. While you are in input mode, you can enter text on the keyboard in the same way that you type text on a typewriter.

On an MBCS AIX system, **vi** allows you to enter and revise Japanese text.

Like all the standard programs of the AIX system, you start **vi** by entering the program name in ASCII characters. You must also enter the **vi** subcommands and option flags in ASCII characters. However, you can enter text into a file as Japanese characters, such as Hiragana, Katakana, Romaji, and Kanji.

Most of the **ex** editor's commands can also be executed within **vi**. See **ex** in *AIX Operating System Commands Reference* for more information about **ex** commands.

Subtopics

1.1.2.1 Common vi Functions

These are special function keys which you will use often. Before you begin, you should locate these keys on your keyboard.

Key	Action
PICTURE 1	When you are in command mode, sends commands to the system. To enter a command, type the command and then press the **Enter** (PICTURE 2) key. When you are in input mode, operates the same as the carriage return key on a typewriter. It brings the cursor down one line to the first position on the next line.
ESC	Instructs **vi** to leave text input mode and enter command mode. **ESC** also cancels partially formed commands. This key is useful if you begin to type a wrong command or change your mind.
DEL	Interrupts the editor while it is processing a command. After you press **DEL**, the editor can accept a new command. You may also find this key useful if you enter a wrong command or change your mind.

1.1.2.2 Refreshing the Screen

If the screen image is lost or altered due to a transmission error or if a program other than the editor writes output to your screen, enter **Ctrl-L** to clear the screen and refresh (or redraw) the display.

1.1.2.3 Entering Shell Commands

You can run any AIX command from within **vi** by entering:

:!**cmd**

where **cmd** is the command you want to process.

When processing is finished, press **Enter** to continue. Then the screen is cleared, and the file is redisplayed.

To process more than one shell command, enter:

:sh

Return to **vi** by entering **Ctrl-D**.

If you start **vi** from the C shell rather than the Bourne Shell, you can enter shell commands without terminating **vi** by temporarily suspending your editing session. To do this, enter **Ctrl-Z** as the first character on a line while in command mode. See "Changing Modes" in topic 1.2.2.2 for information on command mode. After suspending **vi**, you can continue your editing session in the foreground or background with the C shell commands **fg** and **bg**, respectively. See *AIX Operating Systems Commands Reference* for more information about these commands.

You must enter all AIX commands and **vi** subcommands in ASCII characters.

1.1.2.4 International Character Support

The **vi** editor can be used in a Japanese locale to enter and edit Japanese text.

Like all standard AIX programs, the **vi** editor is invoked by entering its program name and option flags in ASCII characters. Once the program is running in input mode, the keyboard is shifted, and Japanese characters can be entered in Katakana, Hiragana, or Kanji mode. Whenever a **vi** subcommand must be entered, the keyboard is shifted to ASCII mode, and the subcommand is entered in ASCII characters.

The **vi** editor handles single Japanese characters and whole lines of Japanese characters in the same way it handles English. Japanese characters and lines can be searched for, deleted, and changed. Japanese words, however, cannot be handled the same as English words. Written Japanese does not usually separate words in a sentence by the blank spaces that **vi** uses (together with punctuation) as word delimiters. If the usual Japanese practice of placing no space between words is followed, **vi** must rely on punctuation alone to delimit words. If the cursor is placed at the beginning of a Japanese sentence and the **cw** subcommand is entered in ASCII, **vi** changes everything up to the first punctuation mark or newline character it encounters. Therefore, the **w** subcommands should be used with care when editing Japanese text.

1.2 Chapter 2. Beginning and Ending Editing Sessions

Subtopics

1.2.1 About This Chapter

The editor stores information (programs, memos, and other text documents) as files. When you are working with a file, it is called an editing session.

This chapter explains how to start and end an editing session. You can start an editing session either by creating a new file or by accessing an existing file. This chapter also explains how to add text to a file by entering text input mode.

When you end the editing session, you can either store the file with your new changes or quit the file without storing any changes.

1.2.2 Starting the vi Editor

The most common way to start **vi** is by entering:

vi **filename**

at the $ prompt.

You start **vi** by entering **vi** in ASCII characters. You can enter the file name, however, in Japanese characters. For example:

vi **Japanese-characters**

Note: For additional ways to start **vi**, see "Viewing the Contents of a File" in topic 1.2.6 and "Editing Several Files" in topic 1.4.2.2.

The **filename** can be the name of a new file or the name of an existing file. Use the following guidelines when you name a file:

Give each file a unique name

Choose names that are easy to remember. For example, the file nam **memoform** may mean more to you than **form1**.

Make file names 1 to 14 characters long

Use letters, numbers, underscores, and other characters that do no have a special meaning to the shell. In general, you should avoid using the following:

* (asterisk)
\ (backslash)
(space)
- (minus)
& (ampersand)
; (semicolon)
? (question mark)
((left parenthesis)
) (right parenthesis)
[(left bracket)
] (right bracket)
{ (left brace)
} (right brace)
! (exclamation point, in **csh**)

Use capitalization and characters carefully. Each time you want t use a file, you must type the file name exactly as you created it. These are all unique file names: **chapter1, CHAPTER1, and Chapter1**.

When you start **vi**, you are in command mode, and the cursor is in the upper left-hand corner of the screen. If you are creating a new file, **vi** displays lines consisting only of tildes (~) along the left side of the screen and the file name at the bottom of the screen. The tildes indicate that you can enter text and are not part of the file. They indicate lines on the screen only and disappear when you enter text. If you are editing an existing file, **vi** displays the contents of the file and tells you the name of the file and the number of lines and characters in that file in a line across the bottom of the screen.

Here is an example of a typical file display.

```
Marigolds are annual plants.  They are fast growing and quick to
produce plentiful flowers.

Many gardeners use marigolds as border or bedding plants for
continuous summer color.  Marigolds also make excellent cut
flowers.
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~

"marigolds"  5 lines, 223 characters
```

The text within the file can be ASCII characters, as shown, or Japanese characters, such as Katakana, Hiragana, Romaji, and Kanji. You can mix Japanese and ASCII characters within the same file.

Note: If you want to learn how to use **vi** by working with an existing file, make a copy of it with the following AIX command:

```
cp oldname newname
```

You can use the copy of this file for practice while you are learning the editor without affecting the contents of the original file.

Subtopics
1.2.2.1 Moving the Cursor
1.2.2.2 Changing Modes
1.2.2.3 Adding Text
1.2.2.4 Saving Changes to a File

1.2.2.1 Moving the Cursor

Since **vi**'s editing commands add text or make changes to the file in relation to the position of the cursor, you need to know how to move the cursor in the file. To move the cursor, you must be in command mode and in a file that contains text. Use the following commands for basic cursor movement. For additional ways to move and scroll through a file, see "More Ways to Move through a File" in topic 1.4.4.

Key	**Moves Cursor:**
or **h**	One space to the left
or **l**	One space to the right
or **k**	Up one line (in the same column)
-	Up one line, to the first column
or **j**	Down one line (in the same column)

+ or PICTURE 3 Down one line, to the first column.

You can also use the spacebar and backspace key to move the cursor forward or backward, and you can precede a cursor movement command with a number to move more than one character or line. For example, the command **6j** moves the cursor down six lines. Most **vi** commands can be preceded by a number to change the effect of the command.

If the cursor reaches the top or bottom of the screen, the text scrolls up or down (if possible) to bring one line at a time into view; the cursor remains on the first or last line of the screen.

1.2.2.2 Changing Modes

When you start **vi**, you are in command mode. In command mode, you enter **vi** commands. In input mode, you can type text.

The following **vi** commands: **i**, **I**, **a**, **A**, **o**, and **O** bring you into input mode. You can type one of these commands at any place on the screen, depending upon where you want to enter text. You will not see the command you type, but the text you type after typing the command appears on the screen.

To end text insertion and move back into command mode, press **ESC**.

You must enter **vi** subcommands in ASCII characters. Once you are in input mode, you can enter text in ASCII characters, Japanese characters, or a combination of the two.

1.2.2.3 Adding Text

You can add text on a line before or after the cursor, and you can insert new lines of text in a file. Use the following commands to add text in your file:

Command	Action
i	Inserts text before the cursor. Everything to the right of the cursor is pushed to the right.
I	Inserts text before the first nonblank character on the current line.
a	Inserts (appends) text after the cursor. Everything to the right of the cursor is pushed to the right.
A	Adds text to the end of the current line.
o	Opens a line after the current line. The cursor is located at the beginning of the new line, where you can begin to enter text.
O	Opens a line before the current line.

Whenever you are adding text, you can enter several lines of input or just a few characters. Use the **Enter** key to move to the next line while you are adding text if you do not want the text to wrap around to the next line.

You can also make use of the **wrapmargin** option, which automatically moves your text to the next line as you type. See the description of the **wrapmargin** option in "Editor Options and EXINIT Profile" in topic 1.5.5 for information on how to make use of this option.

Subtopics
1.2.2.3.1 Correcting Mistakes While Adding Text

1.2.2.3.1 Correcting Mistakes While Adding Text

To correct mistakes while you are adding text, use the following commands:

Command	Action
Ctrl-H	Backspaces one character at a time so you can type over the text. Some Japanese characters are displayed in one column on the screen; others require two columns. **vi** always moves one full character each time you press the backspace key sequence.
Ctrl-W	Backspaces one word at a time so you can type over the text. The **vi** editor recognizes a word as any character sequence separated from other characters by a space (created with the spacebar, tab key, or carriage return key) or by ending punctuation. Use of the word measurement for Japanese text can have unexpected results as Japanese words are not usually separated by a space, and **vi** will judge a word by ending punctuation only.
\	Quotes the erase characters.
Ctrl-D	Moves back to the previous **autoindent** stop; works on a new line immediately after the supplied **autoindent**. The **autoindent** option automatically indents a new line. See the **autoindent** option in "Editor Options and EXINIT Profile" in topic 1.5.5 to find out more about this option.
^Ctrl-D	Ignores the **autoindent** for the current line only. The indent is restored on the next line. The **autoindent** option automatically indents a new line. See the **autoindent** option in "Editor Options and EXINIT Profile" in topic 1.5.5 to find out more about this option.
0Ctrl-D	Ignores the **autoindent** for all lines. The **autoindent** option automatically indents a new line. See the **autoindent** option in "Editor Options and EXINIT Profile" in topic 1.5.5 to find out more about this option.
Ctrl-?	Terminates a **Ctrl-D** command or terminates input mode.
Ctrl-V	Allows you to type nonprinting characters or control characters. For example, you may want to enter printer control codes in your file to control the printing process. If you want to underline a word, for example, enter **Ctrl-V** with the codes that cause the printer to start and stop underlining before and after the appropriate word. (See *Managing the AIX Operating System* if you want more information on printer control codes.) When you enter a **Ctrl-V**, a circumflex (^) character appears on the screen. This indicates that a nonprinting character is to follow. **Note:** You cannot enter the null character (**Ctrl-@**) or the line feed character (**Ctrl-J**). You can, however, insert any other character after entering **Ctrl-V**.

You can enter more than one **Ctrl-H** or **Ctrl-W** to backspace over more than one character or word at a time. You can also use the backspace key to backspace one character at a time; and if you use the number sign (#) as your erase character in the shell, it works the same as **Ctrl-H** in **vi**. If you need to type a number sign (#) you must precede it with a backslash (\) just as you would if you were working in the shell. You can only backspace over the current line. To make corrections to text preceding or following the current line, use the commands described in Chapter 3, "Correcting and Deleting Text."

1.2.2.4 Saving Changes to a File

The **vi** editor does not directly modify the file that you are editing. Instead, it makes a copy of this file in a ***buffer***, which is a portion of storage that holds the file temporarily. If you are editing an existing file, you do not affect its contents until you save (or write) the changes. To save all or part of a file, but not leave **vi**, use the following commands:

Note: When you type a command starting with a colon (:), the cursor automatically moves to the last line of the screen; and the command appears as you type it. You must press **Enter** to process the command.

Command	Action
:w	Saves the changes and remains in **vi**. This command is useful if you are editing for a long time and want to minimize the amount of data that could be lost if the system failed. During a long editing session, it is a good practice to periodically save your changes by using the **:w** command.
:w filename	Saves the changes in the file specified by **filename**. This command is useful if you want to have two versions of a file: the original version and a revised version.
:x,yw!	Saves only lines **x** through **y**. For example, the command **:2,9w!** stores lines 2 through 9. To use this command, you need to know the line numbers you want to save. You can find out what line numbers you want to save in several ways: You can set line numbering on by changing the **number** option. See the **number** option in "Editor Options and EXINIT Profile" in topic 1.5.5 for information on how to change this option. You can use the command **Ctrl-G** to find out the number of the current line. See "Finding Out the Line Number of the Current Line" in topic 1.2.3 for more information on how to use **Ctrl-G**. You can use the concept of relative line numbering where dot (**.**) represents the current line. Therefore, you can use the command **.,±n** where **.** represents the current line and **n** represents the number of lines to save following or preceding the current line. For example, the command **.,+5w!** saves the current line and the next five lines.
:x,yw filename	Saves only lines **x** through **y** in the file specified by **filename**. For example, the command **:2,9w memo** saves lines 2 through 9 in a file named memo. If you specify a file name of an existing file when saving changes, the following message is displayed: Use w! to overwrite an existing file. Retype the command specifying a new file name; or if you want to overwrite the file (have the contents of the current file replace the contents of the specified file), retype the command adding the exclamation mark (!), for example, **:2,9w! memo**.

You must enter all **vi** subcommands in ASCII characters. However, you can enter the *filename* parameter in Japanese characters.

1.2.3 Finding Out the Line Number of the Current Line

To find out the line number where the cursor is located, enter: **Ctrl-G**.

The following message is displayed:

"**filename**" line **x** of **y** --z%--

where:

filename is the name of the file you are editing.
x is the current line number.
y is the number of lines in the file.
z% is the percentage of the way through the file where the cursor is located.

This command is useful to determine where you are in the file and, therefore, can help you determine line numbers if you want to save only certain lines of the file. See "Saving Changes to a File" in topic 1.2.2.4 for information on how to save only certain lines in the buffer.

1.2.4 Ending vi

You can end **vi** and save all the changes, or you can end **vi** without saving any of the changes. When you end **vi**, you return to the shell prompt. Use the following commands to end **vi**:

Command	Action
ZZ	Saves the changes and ends **vi**. The **ZZ** command is a quick way to save the changes and end **vi** as it does not require you to press **Enter** to process the command.
:wq	Saves the changes and ends **vi**.
:q	Used after a **:w** command has been used to save the changes; ends **vi**. Also ends **vi** if you have not made any changes to the file. If you have made changes which you have not saved, **vi** displays a warning message and does not quit.
:q!	Ends **vi** without saving any changes.

1.2.5 Recovering Lost Files

If the system malfunctions (or "goes down"), you can recover the work you were doing to within a few changes. To recover lost files:

1. Change to the directory you were in when the system malfunctioned. See *Using the AIX Operating System* if you need information on how to change directories.

2. Enter:

   vi -r **filename**

   where **filename** is the name of the file you were editing.

vi recovers your file to within the last few changes.

You can get a listing of the files that **vi** recovered for you by entering:

```
vi -r
```

Notes:

1. The command **vi -r** does not always list all saved files; however, any file can be recovered even if it is not listed as long as you know the name of the file.

2. If there is more than one copy of a particular file saved, **vi** recovers the most recent version of the file each time you use **vi -r filename**. Therefore, you can recover an older saved copy by first recovering the newer copies.

You must enter the **vi** command and its option flags in ASCII characters. The *filename* parameter can be ASCII characters or Japanese characters.

1.2.6 Viewing the Contents of a File

You may want to look at a file but not make changes. To view a file, start **vi** by entering the following command at the $ prompt:

```
view filename
```

While in **view** (or "read-only" mode), **vi** does not allow you to make any changes to a file. Use the **:q** command to exit the file.

You must enter all **vi** subcommands in ASCII characters. However, you can enter the *filename* parameter in Japanese characters.

1.3 Chapter 3. Correcting and Deleting Text

Subtopics

1.3.1 About This Chapter

This chapter explains basic ways to make changes to text. For example, you can correct or delete a character, a word, or an entire line. You can also undo any change you make.

1.3.2 Making Corrections

There are many commands to make changes to your text depending upon the results you want to achieve. Remember that commands make changes relative to the position of the cursor. Therefore, before you use a command you must first move the cursor to the appropriate place in the file. If you are editing a large file, you may find it useful now to learn about scrolling and other ways to move the cursor through the file more quickly and efficiently. See "More Ways to Move through a File" in topic 1.4.4 for this information.

Subtopics

1.3.2.1 Correcting Characters

To correct characters, use the following commands:

Command	Action
x	Deletes a character at the cursor position.
rc	Replaces a character at the cursor position with **c**.
sstringESC	Replaces a character at the cursor position with **string**.
dfc	Deletes all characters including character **c** to the right of the cursor.
dFc	Deletes all characters including character **c** to the left of cursor.
dtc	Deletes all characters to the right of the cursor up to but not including character **c**.
dTc	Deletes all characters to the left of the cursor up to but not including character **c**.

You can precede the **x**, **r**, or **s** commands with a number to increase their effect. For example, the command **3x** deletes three characters (to the right of the cursor); and the command **3rc** replaces three characters with character **c**.

You can also use the commands **fc** and **Fc** to simply find character **c** to the right and left of the cursor respectively. The **;** command repeats the last find operation.

Note: Commands which end by pressing **ESC** bring you into input mode. Text insertion is ended, therefore, by pressing **ESC**.

You must enter all **vi** subcommands in ASCII characters. Text represented by the *c* and *string* parameters can be ASCII or Japanese characters.

1.3.2.2 Correcting Words

To correct words, use the following commands:

Command	Action
dw	Deletes a word including the space to the right of the cursor.
db	Deletes a word including the space to the left of the cursor.
cwstringESC	Changes a word to the right of the cursor to **string**. If you replace a word with fewer characters than the original word, the extra characters from the original word are automatically erased when you press **ESC**. If you replace a word with more characters than the original word, the remainder of the line is automatically pushed to the right.

You can also precede these commands with a number to increase their effect. The command **4dw**, for example, deletes four words to the right of the cursor.

The **vi** editor recognizes a word as any character sequence separated from other characters by a space (created with the spacebar, tab key, or carriage return key) or by ending punctuation. Use of the word measurement for Japanese text can have unexpected results as Japanese words are not usually separated by a space, and **vi** will judge a word by ending punctuation only.

1.3.2.3 Correcting Lines

To correct an entire line, use the following commands:

Command	Action
dd	Deletes the current line.
dL	Deletes the current line through the last line on the screen.
d/string	Deletes all lines until **string**. The **/string** part of this command is a search command which instructs **vi** to find the text that you specify. Search commands are described in "Finding Specific Character Strings in a File" in topic 1.4.6. When you type the slash (**/**), the cursor automatically moves to the last line on the screen and the text you specify appears on this line. When you have correctly typed the text to be found, press **Enter** to process the command.
SstringESC	Substitutes **string** for the current line. The cursor automatically moves to the first position on the current line, and the current line is erased. Then, the text you type replaces the current line.
ccstringESC	Same as the above.

You can also delete or change more than one line by preceding these commands with a number. For example, the command **5dd** deletes five lines, the command **3dL** deletes all lines from the current line to three lines from the bottom of the screen, and the command **4ccstringESC** changes four lines to **string**.

1.3.2.4 Correcting Sentences, Paragraphs, and Sections

To delete sentences, paragraphs, and sections, use the following commands:

Command	**Deletes:**
d(	Entire sentence to the left of the cursor.
d)	Entire sentence to the right of the cursor.
d{	Entire paragraph to the left of the cursor.
d}	Entire paragraph to the right of the cursor.
d[[	Entire section to the left of the cursor.
d]]	Entire section to the right of the cursor.

For example, the command **d)** deletes the remainder of the sentence from the location of the cursor.

If you put the cursor at the beginning of the sentence, paragraph, or section, **vi** deletes the previous sentence, paragraph, or section. If you put the cursor at the end of the sentence, paragraph, or section, **vi** deletes the next sentence, paragraph, or section.

Note: A sentence is defined by ending punctuation: a period (.), a question mark (?), or an exclamation point (!). A paragraph is defined by a blank line, or you can redefine a paragraph in the EXINIT profile. A section is defined in the EXINIT profile. See the **paragraph** and **section** options described in "Editor Options and EXINIT Profile" in topic 1.5.5 to find out how to redefine paragraphs and sections.

1.3.3 Reexecuting the Last Command

You may want to repeat the last change command. The **.** (dot) command allows you to do this without having to type the command again.

1.3.4 Undoing a Change

If you change your mind and want to undo what you have done, use the following commands:

Command	Action
u	Undo the last change.
U	Undo all changes made to the current line.

The undo command lets you reverse only the last change. If you enter two undo commands, you would undo the last change and then undo the undo command, which would restore the change again.

1.3.5 Recovering Deleted Lines

If you delete a number of lines and change your mind, you can recover the last nine deleted lines by using the following commands:

Command	Action
"np	Recovers the text in buffer number **n** and puts it (quotation mark) after the cursor.
"nP	Recovers the text in buffer number **n** and puts it (quotation mark) before the cursor.

vi saves the last nine deleted blocks of text in a set of numbered buffers 1 through 9. Since the buffer numbered 1 contains the most recent deletion, the command **"1p** places the last text deleted after the cursor. If this does not recover the text you want, use the **u** command to undo the change and then use the **.** (dot) command to repeat the last change command. When the last change command refers to a buffer (the quote mark (") indicates that a buffer name is to follow), the **.** (dot) command increases the buffer number by one before repeating the command. Therefore, a sequence of commands like **"1pu.u.** recovers the text in the first buffer (the most recent deletion), **u** deletes it, **.** recovers the text in the second buffer, **u** deletes it, and **.** recovers the text in the third buffer. You can stop after any **.** (dot) command to keep what has been recovered, and you can eliminate the undo command (**u**) to recover the text in any or all of the buffers.

1.4 Chapter 4. Advanced Tasks

Subtopics

1.4.1 About This Chapter

This chapter tells you about more advanced editing techniques that will make editing easier for you especially if you edit large files or edit for long periods.

This chapter tells you how to edit more than one file in an editing session and how to run system commands on a file. This chapter also explains more efficient ways to move through a file and how to revise a file by copying and moving text.

1.4.2 Editing Additional Files

Sometimes you may want to edit more than one file during your editing session. You can do this without ending **vi**.

Subtopics
1.4.2.1 Editing a Second File
1.4.2.2 Editing Several Files
1.4.2.3 Adding Another File to the Current File

1.4.2.1 Editing a Second File

To edit another file during your editing session without ending **vi**, use the following commands:

Command	Action
:e filename	Edits the file specified.
:e + filename	Edits the file specified starting at the end of the file.
:e + n filename	Edits the file specified starting at line **n**. In addition to specifying a line number, **n** can be any **vi** command not containing a space. For example, the command **e +/string memo**, searches the file named memo for **string** and places the cursor at that point in the file. "Finding Specific Character Strings in a File" in topic 1.4.6 explains the search commands.
:e #	Edits the alternate file. You may find this command useful to toggle between files.
:ta tag	Edits the file containing **tag**, beginning at the **tag**. This command is useful if you are editing large programs. To use **:ta**, you must create a data base of function names and their locations. See the **ctags** command in the *AIX Operating System Commands Reference* to find out how to create a data base of function names. To find the same tag again, repeat the **:ta** command without specifying a tag.

If you try to edit a second file without saving the changes you made to the first file, the following message is displayed:

```
No write since last change (:edit! overrides)
```

You must first save the changes with the **:w** command or use the **:e!** command to discard the changes made to the current file before editing the alternate file. This is also true for the **:ta** command if it requires **vi** to switch files.

You must enter **vi** subcommands in ASCII characters. However, you can enter the *filename* parameter in Japanese characters.

1.4.2.2 Editing Several Files

You can edit several files during your editing session by starting **vi** with the list of files you want to edit. To do this, enter:

vi **filename1 filename2 filename3 ...**

where **filename 1, filename2, filename3**, and so forth are the names of the files you want to edit. To move between files, use the following commands:

Command	**Action**
:n	Edits the next file in the list of files.
:n!	Edits the next file in the list of files and discards the changes made to the current file.

Command	**Action**
:n filenames	Specifies a new list of files to be edited.

1.4.2.3 Adding Another File to the Current File

You can add the contents of another file or the output of a shell command into the file you are currently editing. The file or output of the command is added after the current line.

To add the contents of a file to the file you are editing, enter:

```
:r filename
```

where **filename** is the name of the file you want to add to the current file.

To add the output of a command to the current file, enter:

```
:r !cmd
```

where **cmd** is the AIX command whose output you want to add to the current file.

You must enter **vi** subcommands and AIX commands in ASCII characters. However, you can enter the *filename* parameter in Japanese characters.

1.4.3 Filtering a File

Filtering is the process of running a system command on all or part of a file. You can, for example, use a system command to sort lines or reformat portions of a file for a printer. To filter a file, enter:

!**cmd**

where **cmd** is an AIX command.

See the *AIX Operating System Commands Reference* if you want more information on filter commands.

You must enter **vi** subcommands and AIX commands in ASCII characters. However, you can enter the *filename* parameter in Japanese characters.

1.4.4 More Ways to Move through a File

In addition to the basic ways to move through a file (that is, the arrow keys and **h**, **j**, **k**, and **l** commands), **vi** has many other commands to help you move through your file quickly. This section introduces you to commands that move you through a file when you have long distances to move the cursor.

All of the cursor movement commands can be preceded by a number to change the amount the cursor moves.

Subtopics

1.4.4.1 Moving to the Top, Middle, or Bottom of the Screen

To move the cursor to the top, middle, or bottom of the screen, use the following commands:

Command	**Moves Cursor:**
H	To the top line (Home) of the screen.
M	To the middle line of the screen.
L	To the last line of the screen.

You must enter **vi** subcommands in ASCII characters.

Any of these commands can be preceded by a number. For example, the command **4H** places the cursor four lines from the top of the screen; and the command **3L** moves the cursor to the third from the last line on the screen.

1.4.4.2 Moving Up and Down through a File

You can move the cursor one line at a time, several lines at a time, or screens at a time. Use these commands to move the cursor up or down through a file:

Command	**Moves the Cursor:**
Ctrl-Y	Up (or backward) one line.
Ctrl-E	Down (or forward) one line.
Ctrl-U	Up (or backward) one-half screen.
Ctrl-D	Down (or forward) one-half screen.
Ctrl-B	Up (or backward) one screen.
Ctrl-F	Down (or forward) one screen.

Moving through a file a screen at a time is also known as ***scrolling***. Scrolling is like turning the pages of a book.

If you are trying to read through the text of the file, you may find it easier to use **Ctrl-U** and **Ctrl-D** in that it is easier to read the file while scrolling is taking place, and it is easier to keep track of the location of the cursor. The **Ctrl-B** and **Ctrl-F** commands give you only a small amount of context, providing two lines of overlap between screens.

Any of these commands can be preceded by a number to change the effect of the command. For example, the command **3Ctrl-F** moves the cursor down three screens; and the command **5Ctrl-E** moves the cursor up five lines.

You must enter **vi** subcommands and AIX commands in ASCII characters. However, you can enter the *filename* parameter in Japanese characters.

1.4.4.3 Moving across a Line

To move the cursor across a line of text, use the following commands:

Command	Moves cursor:
(^)	To the beginning of the current line (to the first nonblank space)
$	To the end of the current line
n|	To the column number specified by **n**. For example, the command **25|** moves the cursor to column number 25.

To move the cursor between words *including punctuation* (that is, punctuation is considered to be a word), use the following commands:

Command	Moves cursor:
e	To the end of current word
w	To the beginning of the next word
b	To the beginning of the previous word.

To move the cursor between words *ignoring punctuation* (that is, punctuation is skipped), use the commands below:

Command	Moves cursor:
E	To the end of the current word
W	To the beginning of the next word
B	To the beginning of the previous word.

The scope of these commands extends beyond the end of a line. For example, if the cursor is located on the last word in a line and you enter **w**, the cursor moves to the beginning of the first word on the next line. Any of these commands can be preceded by a number to change the effect of the command. For example, the command **6w** moves the cursor forward six words.

The **vi** editor recognizes a word as any character sequence separated from other characters by a space (created with the spacebar, tab key, or carriage return key) or by ending punctuation. Use of the word measurement for Japanese text can have unexpected results as Japanese words are not usually separated by a space, and **vi** will judge a word by ending punctuation only.

1.4.4.4 Moving to Sentences, Paragraphs, and Sections

To move between sentences, paragraphs, and sections, use the following commands:

Command	**Moves Cursor:**
)	To the beginning of the next sentence.
(	To the beginning of the preceding sentence.
}	To the beginning of the next paragraph.
{	To the beginning of the preceding paragraph.
]]	To the beginning of the next section.
[[	To the beginning of the preceding section.

These commands can also be preceded by a number to change the effect of the command. For example, the command **3)** moves the cursor forward three sentences.

1.4.4.5 Moving to a Specific Line

To move the cursor to a specific line, use the following commands:

Command	**Moves Cursor:**
nG	To line number **n**. For example, **6G** moves the cursor to the sixth line.
G	To the end of the file.

You must enter **vi** subcommands and AIX commands in ASCII characters. However, you can enter the *filename* parameter in Japanese characters.

1.4.4.6 Returning the Cursor to Its Previous Location

To return the cursor to its previous location, use the following commands:

Command	**Returns Cursor:**
`` (two grave accents)	To the previous location
'' (two apostrophes)	To beginning of the line where the cursor was previously located.

1.4.5 Marking a Specific Location in a File

You can mark any location in a file with a single-letter tag and return to this mark later by naming the tag. To mark a location in the file, move the cursor to the appropriate place in the file and enter:

m**x**

where **x** is any letter you want, such as "a". This letter only identifies the mark you put in the file.

You can later return the cursor to this mark by entering:

Command	**Moves Cursor:**
`x (A grave accent)	To the location of the mark **x**
'x (An apostrophe)	To the beginning of the line containing the mark **x**.

You can also use change commands in combination with a mark to make changes to your file. For example, the command **d`x** deletes all lines from the current line until the line containing the mark **x**; the command **d'x** deletes all lines from the current line up to the exact location of the mark **x**.

Note: A mark is effective for the current editing session only. That is, when you edit another file or end **vi**, the mark disappears.

You must enter **vi** subcommands and AIX commands in ASCII characters. However, you can enter the *filename* parameter in Japanese characters.

1.4.6 Finding Specific Character Strings in a File

Search commands move the cursor to a specific character string. Character strings are words or text that you specify. To locate a character string, use the following commands:

Command	**Searches:**
/string	Forward to beginning of the next occurrence of **string**. For example, the command **/enclosed are** searches forward through the file and places the cursor under the "e" in "enclosed are".
?string	Backward to the preceding occurrence of **string**.
/string/ +n	Forward for **string** and puts the cursor **n** number of lines after the line containing **string**.
/string/ -n	Forward for **string** and puts the cursor **n** number of lines before the line containing **string**.
?string? +n	Backward for **string** and puts the cursor **n** number of lines after the line containing **string**.
?string? -n	Backward for **string** and puts the cursor **n** number of lines before the line containing **string**.

When you type a command preceded by a slash (/) or a question mark (?), the cursor moves to the last line on the screen. The slash (/) or question mark (?) followed by the **string** that you specify appears on this line. You type the **string** you want **vi** to find and then press **Enter**.

You must enter **vi** subcommands in ASCII characters. The text represented by the *string* parameter can be Japanese characters.

The search goes forward to the end of the file (or backward to the beginning of the file) and, if necessary, wraps around past the end (or beginning) of the file until the character string you specify is found. If you want to prevent **vi** from automatically wrapping around during searches, you can change the **wrapscan** option. For information on how to disable the wraparound feature, see the **wrapscan** option in "Editor Options and EXINIT Profile" in topic 1.5.5.

When you issue a search command, **vi** searches for an exact match for the character string, including uppercase and lowercase letters. If you want the search to ignore case, you can change the **ignorecase** option. For information on how to change this option, see the **ignorecase** option in "Editor Options and EXINIT Profile" in topic 1.5.5.

If the string you are searching for is not in the file, **vi** prints a message on the last line of the screen; and the cursor returns to its initial position.

Subtopics
1.4.6.1 Extended Pattern Matching
1.4.6.2 Reexecuting a Search Command

1.4.6.1 Extended Pattern Matching

In addition to specifying a fixed character string for **vi** to search for, you can construct a string with special pattern-matching characters. This string is called a ***regular expression***. Using a regular expression to locate text in your file requires practice and may, at first, seem much more difficult than just specifying a fixed character string. Regular expressions, however, give you more flexibility when you are trying to locate specific text.

The following characters are used to construct a regular expression:

Command	**Action:**
(^)	Finds a match only at the beginning of a line. For example, the command **/^mail** finds the string **mail** only at the beginning of a line.
$	Finds a match only at the end of a line. For example, the command **/mail$** finds the string **mail** only at the end of a line.
.	Matches any character.
<	Finds a match at the beginning of a word.
>	Finds a match at the end of a word.
[string]	Finds a match for any single character in **string**.
[(^)string]	Finds a match for any single character not in **string**.
[x-y]	Finds a match for any character between **x** and **y**.
*****	Finds a match for zero or more repetitions of the preceding character. For example, the command **/ab*cd** matches each of the following strings: **acd**, **abcd**, **abbcd**, and **abbbcd**.

Because these characters have a special meaning, you must precede them with a backslash (\) if you want to type them as "ordinary" characters.

You must enter **vi** subcommands in ASCII characters. The text represented by the *string* parameter can be Japanese characters.

If you do not want to use this extended pattern matching feature, you can change the **magic** option. For information on how to change this option, see the **magic** option in "Editor Options and EXINIT Profile" in topic 1.5.5. If you change this option, only the ^ and $ characters are recognized as special characters, but you can access the special meaning of the other characters by preceding the characters with a backslash (\). If you change this option, you must also precede a slash (/) and a question mark (?) with a backslash (\) in order to issue a forward or backward search.

1.4.6.2 Reexecuting a Search Command

You may want to repeat a search for the same character string. To reexecute the last search command without having to retype the command and the entire string, use the following commands:

Command	**Repeats:**
n	The last search for **string** in the same direction as the previous search.
N	The last search for **string** in the opposite direction as the previous search.

1.4.7 Copying and Moving Text

You may find it useful to copy and move text in addition to using the basic editing commands to correct and delete text. To copy text, use the following commands:

Command	Action
n"cyx	Copies ("yanks") the specified text into a buffer where **n** represents a number, **c** represents the name of the buffer (can be any letter a through z), and **x** represents the text (for example, w for word). Therefore, the command **5"ayw** copies five words into the buffer named a. If you do not specify a buffer name, **vi** copies the text into an unnamed buffer. Therefore, the command **5yw** copies five words into an unnamed buffer.
Y	Copies the current line into the buffer. You can precede the **Y** command with a count. For example, the command **3Y** copies three lines into the buffer.
P	Puts the contents of the buffer before the cursor. If you specified a buffer name when you saved text, use the command **"cP** to put the contents of the buffer named **c** before the cursor.
p	Puts the contents of the buffer after the cursor. If you specified a buffer name when you saved text, use the command **"cp** to put the contents of the buffer named **c** after the cursor.

The **y** and **Y** commands are useful if you want the same text to appear in another part of the same file or in a different file without having to retype the text. The object of a copy command (**y** or **Y**) can also be a search command. For example, the command **y/string** copies the text from the location of the cursor up to the first occurrence of the character string that you specify; and the command **y/string/ -3** copies the text from the location of the cursor up to 3 lines before the line containing the string.

If you want to move text from one part of the file to another or from one file to another, you need to delete the text you want moved and then use the **P** or **p** commands to put the text before or after the cursor. You can precede a delete command with the name of the buffer in which the text is to be stored as in **"a5dd** which deletes five lines into the buffer named **a**. Move the cursor to the appropriate place in the file and use a **P** or **p** command to put them back into the file. If you do not specify a buffer name when copying or moving text, **vi** places the text in an unnamed buffer. You can copy or move the text within the same file by using a put command (**P**, or **p**).

The unnamed buffer is lost, however, when you change files. Therefore, to copy or move text from one file to another, you should do the following:

1. Copy or delete the text into a named buffer.

2. Enter:

   :e **filename**

   where **filename** is the name of the file where you want to put the contents of the buffer.

3. Move the cursor to the appropriate place in the file and enter:

   "cp

   or

   "cP

   where **c** represents the name of the buffer that holds the copied or deleted text.

Note: Before you can switch files in an editing session, you must either save the changes to the current file or discard the changes. See "Editing a Second File" in topic 1.4.2.1 if you need information on how to do this.

1.5 Chapter 5. Editor Defaults and Macros

Subtopics

1.5.1 About this Chapter

This chapter describes the various default settings for **vi** and describes how to tailor an editing session to meet your needs by changing these defaults or options. You can make a temporary change (which affects only the current editing session) or a permanent change (which affects every editing session). This chapter also describes the macro and abbreviation facilities.

1.5.2 Editing Programs

The editor has a number of commands for editing programs. What most distinguishes the editing of programs from the editing of text is the desirability of maintaining an indented structure in the body of the program.

The following **vi** options are useful for editing programs. A complete list of all **vi** options is described in "Descriptions of Editor Options" in topic 1.5.5.2.

Subtopics
1.5.2.1 Autoindent
1.5.2.2 Shiftwidth: Setting Boundaries
1.5.2.3 Matching Expression Limiters
1.5.2.4 Commands for Editing LISP

1.5.2.1 Autoindent

The **autoindent** option helps generate correctly indented programs.

To use this option, enter the following command:

```
:set ai
```

The **autoindent** automatically indents a new line to align it with the previous line. If you are using **autoindent** you can backtab over the indent which it supplies by entering **Ctrl-D**. **Ctrl-D** is described in "Correcting Mistakes While Adding Text" in topic 1.2.2.3.1. **Ctrl-D** backs up to a **shiftwidth** boundary and works only immediately after the supplied **autoindent**.

When you are using **autoindent** you may want to place a label at the left margin of a line. To do this, enter:

```
(^)Ctrl-D
```

The editor will move the cursor to the left margin for one line and restore the previous indent on the next.

If you wish to end the indent and not restore it on the next line, enter:

```
0Ctrl-D
```

1.5.2.2 Shiftwidth: Setting Boundaries

Each time you type **Ctrl-D** you back up one position, normally to an 8-column boundary. You can change this amount by changing the **shiftwidth** option. To change the **shiftwidth** option, enter the following command:

```
:set sw=n
```

where **n** is a number.

For example, the command **:set sw=4** changes the the **shiftwidth** option to a 4-column boundary.

To shift lines in the program left and right, use the following operators:

Operator	Action
< <	Shifts one line left.
> >	Shifts one line right.
<L	Shifts the rest of the lines on the screen left.
>L	Shifts the rest of the lines on the screen right.

1.5.2.3 Matching Expression Limiters

If you have a complicated expression and want to see how the parentheses match, put the cursor at the left or right parenthesis and enter **%**. This shows you the matching parenthesis. This works also for braces ({}), and brackets ([]).

If you are editing C programs, you can use the [[and]] commands to advance or return to a line starting with a {, that is, to move a function declaration at a time. When]] is used with an operator, it stops after a line which starts with }; this is sometimes useful with the **y]]** command.

1.5.2.4 Commands for Editing LISP

If you are editing a **LISP** program, set the option **lisp** by entering:

```
:set lisp
```

This changes the (and) commands to move backward and forward over s-expressions. The { and } commands are like (and) but don't stop at atoms. These can be used to skip to the next list or to skip through a comment quickly.

The **autoindent** option works differently for **LISP** in that it supplies the indent to align at the first argument to the last open list. If there is no such argument, then the indent is two spaces more than the last level.

Another option which is useful for typing in **LISP** is the **showmatch** option. Set the **showmatch** option by entering:

```
:se sm
```

When you type an expression beginning with a left parenthesis (and end it with a right parenthesis), the cursor moves temporarily from the right parenthesis to the left parenthesis. This happens only if the matching left parenthesis is on the screen. Then, the cursor returns to its previous position.

The **-** operator realigns existing lines as though they had been typed in with the **lisp** and **autoindent** options set. Therefore, the command **-%** entered at the beginning of a function declaration realigns all the lines of the function declaration.

When you are editing **LISP**, the **[[** and **]]** commands advance and return to lines beginning with a left parenthesis (and are useful for dealing with entire function definitions.

1.5.3 Using Macros

The macro facility lets you set up a macro so that when you enter a single keystroke, **vi** acts as though you had entered a longer sequence of keystrokes or commands. It is useful to set up a macro if you type the same sequence of commands repeatedly.

There are two ways to set up a macro:

1. Put the macro in a buffer, such as **x**. Then, type **@x** to invoke the macro. To repeat the last macro, type **@@**.

2. Use the **map** command from **vi** (typically added to your EXINIT profile) by entering:

   :map **x string**

   which maps **x** into **string**.

   There are restrictions:

   - The **x** should be one keystroke (either one character or one function key) since it must be entered within one second (unless the **notimeout** option is set, in which case **vi** waits for you to finish it before it echoes anything).
   - The **x** can be no longer than ten characters.
   - The **string** can be no longer than 100 characters.
   - To get a space, tab, or newline into **x** or **string** you should escape them with two **Ctrl-V**'s. Spaces and tabs inside the **string** need not be escaped.

   For example, to make the **q** key write and exit **vi**, enter the following command:

   :map q :wq**Ctrl-V Ctrl-V Enter Enter**

   Now, whenever you type **q**, it is as though you typed **:wqEnter**.

   From within **vi**, you have to enter two **Ctrl-V**'s to get one; a **Ctrl-V** is needed because without it the **Enter** would end the : command rather than becoming part of the map definition. The first **Enter** is part of the **string**; the second terminates the **map** command.

Macros can be deleted by entering:

:unmap **x**

If the **x** of a macro is **#0** through **#9**, this maps the particular function key instead of the two-character # sequence. Therefore, terminals without function keys can access such definitions; the form **#x** means function key **x** on all terminals (and need not be typed within 1 second). The # character can be changed by using a macro in the usual way:

:map **Ctrl-V Ctrl-V Ctrl-I** #

to use tab, for example. This will not affect the **map** command, which still uses #, but just the invocation from **vi**.

Placing an exclamation point (**!**) after the word map causes the mapping to apply to input mode rather than command mode. Thus, to make **Ctrl-T** the same as four spaces in input mode, enter:

```
:map! Ctrl-T Ctrl-V
```

where is a blank space. The **Ctrl-V** is necessary to prevent the blanks from being taken as white space between the **x** and **string**.

Note: When defining a macro, be careful not to duplicate any existing **ex** or **vi** commands, unless you want to redefine them.

1.5.4 Word Abbreviations

A word abbreviation, which operates in input mode, is a feature similar to a macro, which operates in command mode. A word abbreviation allows you to type a short word and have it expanded into a longer word or words. The commands to set abbreviation and to delete it are:

:abbreviate
:unabbreviate

or:

:ab
:una

These commands have the same syntax as the **map** command. For example:

```
:ab eecs Electrical Engineering and Computer Sciences
```

causes the word **eecs** to be changed into the phrase, **Electrical Engineering and Computer Sciences**. Word abbreviation is different from macros in that only whole words are affected. If **eecs** were typed as part of a larger word, it would be left alone. Also, the partial word is echoed as it is typed. There is no need for an abbreviation to be a single keystroke as it should be for a macro.

1.5.5 Editor Options and EXINIT Profile

The editor has a set of options which control the characteristics of your editing session. These options are described in "Descriptions of Editor Options" in topic 1.5.5.2.

There are three kinds of options: numeric, string, and toggle. Set numeric and string options by entering:

  set **opt=val**

Set and unset toggle options by entering:

  set **opt**
  set no**opt**

To temporarily change the characteristics of your editing session, enter these commands preceded by a colon (:) while you are running **vi**, for example, **:set opt**, where **opt** is the option you want to change. To permanently change the characteristics of your editing session, place these commands (without the preceding colon) in your EXINIT profile.

To get a list of all options that you have changed, enter the following command from within **vi**:

  :set ?

To find out the value of a single option, enter:

  :set **opt**?

To list all options and their values, enter:

  :set all

You can place multiple options on one line, for example:

  :set ai aw nu

Subtopics
1.5.5.1 Editor Default Options: EXINIT
1.5.5.2 Descriptions of Editor Options

1.5.5.1 Editor Default Options: EXINIT

Options set by the **set** command last only for the current editing session. You may want to have certain options set whenever you use **vi**. To do this create a list of commands which are to be run every time you start **vi**. A typical list includes a **set** command and possibly a few **map** commands. Since it is advisable to put these commands on one line, separate them with the | character; for example:

```
set ai aw terse|map @ dd|map # x
```

The **set** command establishes the **autoindent**, **autowrite**, and **terse** options. The first **map** command makes @ delete a line, and the second **map** command makes # delete a character (see "Using Macros" in topic 1.5.3 for information on macros). Place this string in the variable EXINIT in your environment.

Or, in the AIX shell, put these lines in the **.profile** file in your home or working directory:

```
EXINIT=set ai aw terse|map @ dd|map # x
export EXINIT
```

Or put this line in the **.exrc** in your home directory:

```
set ai aw terse|map @ dd|map # x
```

Note: The **.profile** is executed at login, while the **.exrc** file is executed each time your start **vi**. Therefore, the options in **.exrc** override those in **.eprofile**.

1.5.5.2 Descriptions of Editor Options

autoindent,ai(default: **noai**)
Automatically indents lines of text to align with existing text and, therefore, is useful for editing programs.

If you add additional space after an **autoindent**, the following line automatically aligns with the new level of indentation.

To back up one indent amount at a time, press **Ctrl-D**. The back up indent amounts are defined as multiples of the **shiftwidth** option. To ignore the **autoindent** for the current line and restore it on the next line, press **^Ctrl-D**. To ignore the **autoindent** for all lines, press **0Ctrl-D**. The **autoindent** option specially processes these lines and indents the following lines properly. The **autoindent** option ignores a blank line.

The **autoindent** option is not in effect for **global** commands or when the input is not from a terminal.

autoprint,ap(default: **ap**)
Prints the current line after any command that changes the editing buffer, such as, **delete**, **copy**, or **undo** commands. This has the same effect as supplying a trailing **p** to each such command. The **autoprint** option is suppressed in global commands and only applies to the last command in a sequence of commands on a single line.

autowrite,aw(default: **noaw**)
Writes the editing buffer to the file automatically before the **:n**, **:ta**, **Ctrl-^** and **!** commands if the editing buffer has been changed since the last **write** command.

beautify,bf(default: **nobf**)
Causes all control characters except tab, newline, and formfeed to be discarded from the input. A message is displayed the first time a backspace character is discarded. The **beautify** option does not apply to command input.

directory,dir(default: dir=**/tmp**)
Displays the directory that contains the editing buffer. If this directory in not writable, **vi** exits abruptly when it fails to be able to create its buffer there.

edcompatible,ed(default: **noed**)
Retains global (**g**) and confirm (**c**) subcommand suffixes during multiple substitutions and causes the read (**r**) suffix to work like the **r** subcommand.

errorbells,eb(default: **noeb**)
Error messages are preceded by a bell. If possible, **vi** places the error message in reverse video (white lettering on a black background) instead of ringing a bell. This options may not work on some workstations, such as the PS/2.

flash,fl(default: **fl**)
Flashes error messages. If **nofl** is set, error messages are preceded by a bell. This option may not work on some workstations, such as the PS/2.

hardtabs,ht(default: **ht=8**)
Tells **vi** the distance between the hardware tab stops on your display.

ignorecase,ic(default: **noic**)
Ignores distinction between upper and lower case while searching for regular expressions.

lisp(default: **nolisp**)
Removes the special meaning of **(), {}, [[**, and **]]** and enables the = (formatted print) operator for S-expressions, so you can edit LISP programs.

list(default: **nolist**)
Displays text with tabs and the end of lines marked. Tabs are displayed as ^I and the end of lines as $.

magic(default: **magic**)
Treats the characters ., [, and * as special characters in scans. If **nomagic** is set, only the ( ) and $ retain special meanings; however, special meaning of other characters can still be evoked by preceding the character with a \.

mesg(default: **mesg**)
Causes write permission to be turned off to the terminal if **nomesg** is set.

modeline(default: **nomodeline**)
Runs editor command lines found in the first five and the last five lines of the file. An editor command line may be anywhere in a line. To be recognized as a command line, it must contain a space or a tab followed by the string **ex:** or **vi:**. The command line is ended by a second :. The editor tries to interpret any data between the first and second : as editor commands.

number,nu(default: **nonu**)
Displays lines printed with their line numbers.

optimize,opt(default: **opt**)
Throughput of text is expedited by setting the terminal not to do automatic carriage returns when printing more than one (logical) line of output, greatly speeding output on terminals without addressable cursors when text with leading white space is printed.

paragraphs,para(default: **para=IPLPPPQPP LIbp**)
Defines to **vi** macro names that start paragraphs. Single letter **nroff** macros, such as **.P** must include the space as a quoted character if re-specifying a paragraph.

prompt(default: **prompt**)
Command mode input is prompted for with a colon(:).

readonly (default: **noreadonly**)
Sets permanent read-only mode.

redraw,re(default: **nore**)
The editor simulates (using great amounts of output) an intelligent terminal on a dumb terminal (for example, during

text insertions, the characters to the right of the cursor position are refreshed as each input character is typed). This option is useful only at very high speed.

remap(default: **remap**)
If on, macros are repeatedly tried until they are unchanged. For example, if **o** is mapped to **0**, and **0** is mapped to **I**, then if **remap** is set, **o** maps to **I**, but if **noremap** is set, it maps to **0**.

report(default: **report=5**)
Sets the number of repetitions of a command before a message is displayed. For subcommands that can produce a number of messages, such as global subcommands, the messages are displayed when the command is complete.

scroll(default: **scroll=12**)
Sets the number of lines to be scrolled when you scroll up or down.

sections,sect(default: sect=SHNHH HU)
Defines to **vi** macro names that start sections. Single letter **nroff** macros, such as **.P** must include the space as a quoted character if re-specifying a paragraph.

shell,sh(default: **sh=/bin/sh**)
Defines the shell for **!** or **:!** commands.

shiftwidth,sw(default: **sw=8**)
Sets the distance for the software tab stops used by **autoindent**, the shift commands (> and <), and the input commands (**Ctrl-D** and **Ctrl-T**) to allow **vi** to indent text and move back to a previous indentation.

showmatch,sm(default: **sm**)
Shows the matching open parenthesis (or open brace { as you type the close parenthesis) or close brace }. This option is useful when editing LISP programs.

showmode,smd(default: **nosmd**)
Displays a message to indicate when you are in input mode.

slowopen,slow(terminal dependent)
Postpones updating the display during inserts.

tabstop,ts(default: **ts=8**)
Sets the distance between tab stops when a file is displayed.

taglength,tl(default: **tl=0**)
Sets the number of characters used to recognize a tag. A value of zero (the default) means that all characters are significant.

tags(default: **tags=tags /usr/lib/tags**)
Identifies a path of files to be used as tag files for the **tag** command. A requested tag is searched for sequentially in the specified files. By default, files called **tags** are searched for in the current directory and in **/usr/lib** (a master file for the entire system).

term(default: **$TERM**)
Sets the kind of workstation you are using. The default is

$TERM, where **$TERM** is the value of the shell variable **TERM**. You cannot set **TERM** while you are in **vi**; set **TERM** in the shell.

terse(default: **noterse**)
Allows **vi** to display the short form of messages.

timeout,to(default: **noto**)
Sets a time limit of one second on entry of characters. This limit allows the characters in a macro to be entered and processed as separate characters when **timeout** is set. To resume use of the macro, set **notimeout**.

warn(default: **warn**)
Displays a warning message before the **!** subcommand executes a shell command if this is the first time you have issued a shell command after a given set of changes have been made in the editing buffer but not written to a file.

window,wi(default: **wi=speed dependent**)
Sets the number of lines displayed in one window of text. The default is dependent on the baud rate at which you are operating: 600 baud or less / 8 lines, 1200 baud / 16 lines, higher speeds / full screen minus one.

wrapmargin,wm(default: **wm=0**)
Sets the margin for automatic word wrapping from one line to the next. A value of zero indicates no word wrapping.

wrapscan,ws(default: **ws**)
Allows string searches to wrap from the end of the editing buffer to the beginning.

writeany,wa(default: **nowa**)
Turns off the checks usually made before a **write** command. allowing a write to any file which the system protection mechanism allows.

1.6 Chapter 6. A Summary of vi Commands

Subtopics

1.6.1 About This Chapter

This chapter is a summary of **vi** commands arranged alphabetically by task. Most **vi** commands can be preceded by a number to affect the results of the command. Examples of reasons you may want to do this are:

To change the amount you want to move through a file using any curso movement command.
To change the amount of text you want to change using commands t delete or correct text.

1.6.2 Adding Text to a File

Command	Action
a	Inserts (appends) text after the cursor
A	Adds text to the end of the current line
i	Inserts text before the cursor
I	Inserts text before the first nonblank character on the current line
o	Opens a line after the current line
O	Opens a line before the current line.

1.6.3 Correcting Text

The following sections describe how to make corrections to text in a file.

Subtopics

1.6.3.1 Correcting Mistakes While Adding Text (While in Input Mode)

Command	Action
\	Quotes the erase characters
Ctrl-D	Moves back to the previous **autoindent** stop; works on a new line immediately after the supplied **autoindent**
^Ctrl-D	Ignores the **autoindent** for the current line; the indent is restored on the next line
0Ctrl-D	Ignores the **autoindent** for all lines
Ctrl-H	Backspaces one character at a time
Ctrl-V	Allows you to type non-printing characters or control characters
Ctrl-W	Backspaces one word at a time
Ctrl-?	Terminates a **Ctrl-D** command or terminates input mode.

1.6.3.2 Correcting Characters

Command	Action
x	Deletes a character at the cursor position
rc	Replaces a character at the cursor position with **c**
sstringESC	Replaces a character at the cursor position with **string**
dfc	Deletes all characters including character **c** to the right of the cursor
dFc	Deletes all characters including character **c** to the left of the cursor
dtc	Deletes all characters to the right of the cursor up to but not including character **c**
dTc	Deletes all characters to the left of the cursor up to but not including character **c**.

1.6.3.3 Correcting Words

Command	**Action**
dw	Deletes a word including the space to the right of the cursor
db	Deletes a word including the space to the left of the cursor
cwstringESC	Changes a word to the right of the cursor to **string**.

1.6.3.4 Correcting Lines

Command	**Action**
dd	Deletes the current line
dL	Deletes the current line through the last line on the screen
d/string	Deletes all lines until the first occurrence of **string**
SstringESC	Substitutes **string** for the current line
ccstringESC	Same as the above.

1.6.3.5 Correcting Sentences, Paragraphs, and Sections

Command	**Action:**
d(	Deletes entire sentence to the left of the cursor
d)	Deletes entire sentence to the right of the cursor
d{	Deletes entire paragraph to the left of the cursor
d}	Deletes entire paragraph to the right of the cursor
d[[	Deletes entire section to the left of the cursor
d]]	Deletes entire section to the right of the cursor.

1.6.3.6 Copying and Moving Text

Command	**Action**
n"cyx	Saves the specified text in a buffer where: **n** represents a number (a number is optional) **c** represents the name of the buffer (can be any letter from a through z and is optional; if a buffer name is not specified, **vi** saves the text in an unnamed buffer **x** represents the text (for example, **w** for word)
p	Puts the contents of the buffer after the cursor
P	Puts the contents of the buffer before the cursor
Y	Copies the current line into the buffer.

1.6.4 Editing Additional Files

The following sections describe how you can edit more than one file in an editing session.

Subtopics

1.6.4.1 Adding Another File to the Current File

Command	Action
:r filename	Adds the contents of a file specified by **filename** to the current file; the file is added to the current file after the current line
:r !cmd	Adds the output of a shell command specified by **cmd** to the contents of the current file; the output of the command is added to the current file after the current line.

1.6.4.2 Editing a Second File

Command	Action
:e filename	Edits the file specified by **filename**
:e + filename	Edits the file specified by **filename** starting at the end of the file
:e + n filename	Edits the file specified by **filename** starting at line number **n**
:e #	Edits the alternate file
:ta tag	Edits the file containing **tag**, beginning at **tag**; to use the **:ta** command you must first create a data base of function names and their locations.

1.6.4.3 Editing Several Files

vi filename1 filename2 filename3 ...
Starts **vi** with the names of files you want to edit specified by **filename1**, **filename2**, **filename3**, and so forth

:n Edits the next file in the list of files

:n! Edits the next file in the list of files and discards the changes made to the current file

:n filename1 filename2 filename3 ...
Specifies a new list of files to be edited.

1.6.5 Editing LISP Programs

Command	**Moves Cursor:**
(	To the beginning of the previous s-expression
{	To the beginning of the previous list
)	To the beginning of the next s-expression
}	To the beginning of the next list.

1.6.6 Ending vi

Command	Action
:q	Ends **vi** after a **:w** command has been used to save the changes or ends **vi** if no changes have been made to the file
:q!	Ends **vi** and discards any changes
:wq	Save the changes and ends **vi**
ZZ	Saves the changes and ends **vi**.

1.6.7 Entering Shell Commands

Command	Action
:!cmd	Runs the shell command **cmd** without ending **vi**
:sh	Enters the shell to run more than one shell command; does not end **vi**.

1.6.8 Filtering a File

Command	**Action**
!cmd	Runs the AIX command **cmd** on all or part of a file.

1.6.9 Finding Out the Line Number of the Current Line

Command	**Action**
Ctrl-G	Displays **"filename" line x of y --z%--**, where: **filename** is the name of the file you are editing **x** is the current line number **y** is the number of lines in the file **z%** is the percentage of the way through the file where the cursor is located.

1.6.10 Finding a Specific Character in a Line

Command	Action
fc	Moves the cursor to character **c** to the right of the cursor
Fc	Moves the cursor to character **c** to the left of the cursor
;	Repeats the last find command.

1.6.11 Finding a Specific Character String in a File

Command	**Searches:**
/string	Forward to the beginning of the next occurrence of **string**
?string	Backward to the preceding occurrence of **string**
/string/ ± n	Forward for **string** and puts the cursor **n** number of lines after or before the line containing **string**
?string/ ± n	Backward for **string** and puts the cursor **n** number of lines after or before the line containing **string**.

Subtopics
1.6.11.1 Extended Pattern Matching
1.6.11.2 Reissuing a Search Command

1.6.11.1 Extended Pattern Matching

Command	Action
^	Finds a match only at the beginning of a line
$	Finds a match only at the end of a line
.	Matches any character
<	Finds a match at the beginning of a word
>	Finds a match at the end of a word
[string]	Finds a match for any single character in **string**
[^string]	Finds a match for any single character not in **string**
[x-y]	Finds a match for any character between **x** and **y**
*****	Finds a match for zero or more repetitions of the preceding character.

1.6.11.2 Reissuing a Search Command

Command	**Repeats:**
n	The last search for **string** in the same direction as the previous search
N	The last search for **string** in the opposite direction as the preceding search.

1.6.12 Marking a Specific Location in a File

Command	**Action**
mx	Marks a location in a file, where **x** is any letter you want to identify the mark
`x (grave accent)	Returns the cursor to the mark specified by **x**
'x (apostrophe)	Returns the cursor to the beginning of the line containing the mark specified by **x**.

1.6.13 Moving through a File

The following sections describe how you can move the cursor through a file.

Subtopics

1.6.13.1 Moving Up and Down through a File

Command	**Moves Cursor:**
or **h**	One space to the left
or **l**	One space to the right
or **k**	Up one line (in the same column)
or **j**	Down one line (in the same column)
-	Up one line, to the first column
+ or	PICTURE 4 Down one line, to the first column
Ctrl-D	Down (or forward) one-half screen
Ctrl-F	Down (or forward) one screen
Ctrl-E	Down (or forward) one line
Ctrl-U	Up (or backward) one-half screen
Ctrl-B	Up (or backward) one screen
Ctrl-Y	Up (or backward) one line
H	To the top line (Home) on the screen
L	To the last line on the screen
M	To the middle line on the screen.

1.6.13.2 Moving across a Line

Command	**Moves Cursor:**
^	To the beginning of the current line (first nonblank space)
$	To the end of the current line
n |	To the column number specified by **n**
e	To the end of the current word (considers punctuation as a word)
w	To the beginning of the next word (considers punctuation as a word)
b	Back to the beginning of the last word (considers punctuation as a word)
E	To the end of the current word (skips punctuation)
W	To the beginning of the next word (skips punctuation)
B	Back to the beginning of the last word (skips punctuation).

1.6.13.3 Moving to a Specific Line

G To the end of the file

nG To line number **n**.

1.6.13.4 Moving to Sentences, Paragraphs, and Sections

Command	**Moves Cursor:**
)	To the beginning of the next sentence
(	To the beginning of the preceding sentence
}	To the beginning of the next paragraph
}	To the beginning of the preceding paragraph
]]	To the beginning of the next section
[[	To the beginning of the preceding section.

1.6.13.5 Returning the Cursor to Its Previous Location

Command	**Returns Cursor:**
``(two grave accents)	To its previous location
''(two apostrophes)	To the beginning of the line where the cursor was previously located.

1.6.14 Recovering Deleted Lines

Command	Action
"np	Recovers the deleted line in buffer number **n** and puts it after the cursor; **n** is a number from 1 through 9; the text in buffer number 1 is the most recently deleted text
"nP	Recovers the deleted line in buffer number **n** and puts it before the cursor; **n** is a number from 1 through 9; the text in buffer number 1 is the most recently deleted text.

1.6.15 Recovering Lost Files

Command	**Action**
vi -r	Gives you a list of files that **vi** recovered
vi -r filename	Recovers the file specified by **filename** to within the last few changes.

1.6.16 Refreshing the Screen

Command	**Action**
Ctrl-L	Clears the screen and refreshes (or redraws) the display.

1.6.17 Reissuing the Last Command

Command	**Action**
`.` (dot)	Repeats the last change command.

1.6.18 Saving Changes to a File

Command	**Action**
:w	Saves the changes and remains in **vi**
:w filename	Saves the changes in the file specified by **filename**
:x,yw!	Saves only line numbers **x** through **y**.

1.6.19 Starting vi

Command	**Action**
vi filename	Starts **vi** where **filename** is the name of the file you want to edit
vi filename1 filename2 filename3 ...	Starts **vi** where **filename1**, **filename2**, **filename3**, and so forth is the list of files that you want to edit.

1.6.20 Suspending vi

Ctrl-Z Temporarily suspends your editing session.

1.6.21 Undoing Changes

Command	Action
u	Undo the last change
U	Undo all the changes made to the current line.

1.6.22 Viewing the Contents of a File

Command	**Action**
view filename	Allows you to view the file specified by **filename** but does not allow you to make any changes to the file.

2.0 Part 2. Text Formatting Guide

Subtopics

2.7 Chapter 7. Memorandum Macros

Subtopics

2.7.1 About This Chapter

Memorandum macros are a package of instructions that you can use to format text with the **nroff** and **troff** text formatters.

You cannot use the **mm** macro package to format Japanese text with the **nroff** and **troff** programs as these programs do not support multibyte characters.

The memorandum macros provide a flexible tool for producing many common types of documents. The macros described here should be used for document formatting in most cases because:

You can control and revise the overall style of a document when standard macro package is used.

You can control complex features (such as footnotes and tables o contents) when a standard macro package is used.

You are relieved of the many details involved with using the formatte language directly.

The uses of memorandum macros range from single-page letters to documents of several hundred pages in length--such as user guides, design proposals, and so on. You can use memorandum macros to produce:

Memorandum
Publication
Letter
General documents

The term ***formatter*** refers to either of the text formatting programs **nroff** or **troff**.

Requests are built-in commands recognized by the formatters. Although you seldom need to use these requests directly, this section contains references to some of them. See Chapter 8, "Using nroff and troff" for details.

Macros are named collections of requests. Each macro is an abbreviation for a collection of requests that would otherwise require repetition or complex assembly of request sequences. Memorandum macros supply many macros, and you can define additional ones. Macros and requests share the same set of names and are used in the same way. Macros are called by preceding the macro name (usually two characters) with a period or an apostrophe and placing the macro as the first entry of a line (first characters following a new-line character).

String registers are string variables, each of which is a name for a string of characters. The contents of a string register are called by a string name. String registers are often used for page headers, page footers, and lists. String registers share the pool of names used by requests and macros. You give a string register a value using the formatter ***define string*** request **.ds**. The string value replaces the name when the string is called by preceding the string register name with ***** for one-character names or ***(** for two-character names. For instance, the **DT** string register in the **mm** command normally contains the current date, so when you enter:

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

this could result in the following output:

```
  Today is April 20, 1989.
```

You can initialize the current date by entering:

```
  .ds DT 04/20/89
```

or call a macro designed for that purpose (see "Setting the Date" in topic 2.7.3.4).

Number registers fill the roles of integer variables. They are used for flags, for arithmetic, and for automatic numbering. You can give a number register a value (or change a value) using the **.nr** (number register) request, and reference it by prefacing its name with a **\n** (for one-character names) or **\n(** (for two-character names). For example, the following sets the value of the register **d** to 1 more than that of the **dd** register:

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

See "User-Definable Names" in topic 2.7.11.3 regarding naming conventions for requests, macros, string registers, and number registers. "Part 3. Text Formatting Reference" lists macros, string registers, and number registers defined in memorandum macros and the formatter.

2.7.2 Overview

The following sections describe the design features of the memorandum macro package, the structure segments of an **mm** command, and how to use memorandum macros. It also describes formatting concepts and includes a list of formatter requests.

Subtopics

2.7.2.1 Design Features

The following are design features that are emphasized in the memorandum macro package:

Error recovery. When input is incorrect, the macros attempt to interpret it or recover from the error. Serious errors generate a message describing the error.

Flexibility. Many parameters are provided so you can adapt the output to your needs.

Extendability. You can modify macros for your needs or build your own macros.

- Mnemonic naming conventions are extensively used.
- Guidelines for macro naming are provided for your consideration when expanding your set of macros.
- Macros are constructed consistently.

Device independence.

- A common use of the **mm** command is to print documents on letter quality printers, using the **nroff** formatter. You can use the macros conveniently for both 10- and 12-pitch print.
- You can scan output with an appropriate display.
- The macros are constructed to be compatible with **troff** so you can produce phototypeset output or have your document typeset by a vendor that has a phototypesetting capability. (Since the **mm** command works with both **nroff** and **troff**, you can proofread typed output before vending the typesetting task.)

Macro-level control. The memorandum macros are designed to minimize repetitive typing. For example, if you want a blank line after all level 1 or level 2 headings, you set a specific parameter only once at the beginning of a document, rather than add a blank line after each such heading.

Selectable output style. You can prepare an input text once and produce different output styles by setting a few global flags on the command line or by changing a memorandum type specification in the input file.

2.7.2.2 Structure of a Document

The input for a document to be formatted with the **mm** command is made up of four major segments, any of which can be omitted. If present, they must occur in the following order:

1. **Setup**. This segment sets the general style and appearance of a document. You can control page width, margin justification, numbering styles for headings and lists, page headers and footers, and many other properties of the document. Also, you can add macros or redefine existing ones. You can omit this segment if you are satisfied with the default values. Setup does not produce actual output, but only sets parameters for the rest of the document.

2. **Opening**. This segment includes those items that occur only once at the beginning of a document, such as title, author's name, and date.

3. **Body**. This segment is the actual text of the document. It can be as small as a single paragraph or as large as hundreds of pages. It can have a hierarchy of headings up to seven levels deep. Headings are automatically numbered if you so specify and can be saved to generate the table of contents. Five additional levels of subordination are provided by a set of ***list macros*** for the automatic numbering, alphabetic sequencing, and marking of list items.

   The body can also contain various types of displays, tables, figures, references, and footnotes.

4. **Closing**. This segment contains those items that occur only once at the end of a document. Included here are signature(s) and lists of notations. You can call certain macros 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 of a document.

The existence and size of these four segments vary widely among different memorandum types. Although a specific item (such as date, title, author name(s), and so on) can be typeset in different ways depending on the type of document, there is a uniform way of typing it in. Not all memorandum types use all the information you can provide.

2.7.2.3 International Character Support

Neither **nroff** nor **troff** supports Japanese characters. These programs can format only text consisting of English and European characters.

2.7.2.4 Using Memorandum Macros

This section tells how to use memorandum macros, shows AIX PS/2 Operating System command lines appropriate for various output devices, and describes command line flags for memorandum macros. To use these macros, you enter the **mm** command for **nroff** output, or **mmt** command for **troff** output.

Subtopics

2.7.2.4.1 The mm Command

The **mm** command is used to print documents using **nroff** and memorandum macros; this command calls **nroff** with the **-mm** flag (see "The -mm Flag" in topic 2.7.2.4.3). The **mm** command has flags to specify preprocessing by **tbl** and by **neqn** and for postprocessing by **col**. Any parameters or flags not recognized by the **mm** command are interpreted as flags to **nroff**. The **mm** flags, which can occur in any order but must appear before the file names, are:

-e Calls the **neqn** equation filter.

-t Calls the **tbl** table filter.

-c Calls the **col** output-buffering postprocessor (see **col** in the *AIX Operating System Commands Reference*).

-E Uses the **-e** flag of **nroff** (see topic 2.8.2.1).

-12 Formats for a printer set to a pitch of 12. (Assure that the pitch setting is 12 pitch if the printer has a pitch selection switch.)

-Tterm Selects the display or printer to format for; **term** is the desired terminal type. For example **-T745** causes **nroff** to set type for the IBM PC Graphics Printer. Enter **help term1** for a list of available types. If you give no **-T** option, the value stored in **$TERM** (see *AIX Operating System Technical Reference*) is used if it has a value; otherwise, the default value is **-Tlp**, which is for a nonbackspacing ASCII line printer. Any unknown **-T** option selects **-Tlp**.

2.7.2.4.2 The mmt Command

To call memorandum macros with the **troff** for a phototypesetter, use the **mmt** command. It works much like the **mm** command, but has some different flags and different meanings for the same flags:

-c Calls the **cw** constant-width filter.

-e Calls the **eqn** equation filter.

-t Calls the **tbl** table filter.

-a Uses the **-a** flag of **troff** to create an approximation of output that is in ASCII format.

2.7.2.4.3 The -mm Flag

You can also call the memorandum macro package by including the **-mm** flag as an argument to the **nroff** or **troff** formatter. The **-mm** flag causes the file **/usr/lib/tmac/tmac.m** to be read and processed before any other files, which:

- Defines the memorandum macro
- Sets default values for various parameter
- Prepares the formatter to process the files of input text

2.7.2.4.4 Typical Command Lines

Some typical command lines follow (the parameters are explained under "Setting Number Registers from the Command Line" in topic 2.7.2.4.5).

Plain Text

mm [parms] filename ...

nroff [parms] -mm filename ...

mmt [parms] filename ...

troff [parms] -mm filename ...

Text with Tables

mm -t [parms] filename ...

tbl filename ... | nroff [parms] -mm

mmt -t [parm] filename ...

tbl filename ... | troff [parm] -mm

Text with Equations

mm -e [parms] filename ...

neqn filename ... | nroff [parms] -mm

tbl -TX -mm [parms] filename ...

mmt -e [parms] filename ...

eqn filename ... | troff [parm] -mm

Text with Tables and Equations

mm -t -e [parm] filename ...

tbl filename ... | neqn | nroff [parm] -mm

mm -t -e [parm] filename ...

tbl filename ... | eqn | troff [parm] -mm

Specifying the Printer: When formatting a document with **nroff**, you normally prepare text for output to a specific type of terminal or printing device. Some commonly used terminal types and command lines appropriate for them follow.

Note: The output can require some processing for features that are specific to a given printer such as reverse paper motion (***backscrolling***) or half-line paper motion in both directions.

For further information, see "Setting Number Registers from the Command Line" in topic 2.7.2.4.5.

To specify :

Hewlett-Packard HP26 **x** video display family, enter:
mm -Thp filename ...
or
nroff -mm filename ... | col | hp

Any printer not capable of backscrolling stopping tabs, enter
mm -c filename ...
or
nroff -Tlp -mm filename ... | col -x

IBM PC Graphics Printer (and many popular dot-matrix personal compute printers), enter:
mm -T745 filename ...
or
nroff -Tlp -mm filename ... | col -x

Of course, if you need **tbl** and **neqn** or **eqn**, simply call them as shown in the command line templates under "Typical Command Lines."

Printing Two-Column Output: If you wish to perform two-column processing using **nroff**, you must use **col**. You can simply use the **-c** parameter with **mm** or postprocess the **nroff** output with **col**. **mm -c** handles output processing automatically, but you must do the following if you select postprocessing flags.

1. Specify **-T37** for the terminal type.

2. Do not use the **-h** flag of **nroff**.

3. Filter **col** output through an appropriate terminal filter, such as **745** or supply **col** with appropriate parameters.

Note: **mm** uses the **col** filter automatically for many terminal types.

See "Two-Column Format" in topic 2.7.4.8.1 and "Two-Column Headings" in topic 2.7.4.8.2, for formatting information for two-column output.

2.7.2.4.5 Setting Number Registers from the Command Line

Number registers are used to hold in memory memorandum macro parameter values that control aspects of formatting text. In practice, most of these registers control output style. Many number registers are preset by **nroff** or **troff**. You can set initial values in number registers that have a single character name on the command line; it is useful to set those parameters that should not be permanently embedded within the input text. If you initialize these registers, with the exception of the **P** register, you must set them on the command line (or before the memorandum macro definitions are processed) using the **-r** register flag as a prefix; the flag thus has the form **-rXval**, where **X** is the register and **val** is some value for the register to hold. Only number registers with a single-character name can be set from the command line.

You can set many (but not all) number registers of both single-character and two-character names within text files using **.nr** requests.

The following are number registers that are preset by the formatter:

A Holds a value specifying the format of the **Subject/Date/From** memorandum block of the alternate first-page format that is set by the **.AF** macro (see "Alternate First-Page Format" in topic 2.7.3.6). A flag of the form **-rA1** has the effect of invoking the **.AF** macro without a parameter, suppressing the default company name, the **Subject**, **Date**, and **From** headings, and various default page layout characteristics.

C Holds a value that sets the type of copy (for example, **DRAFT**) that is printed at the bottom of each page. For a flag of the form **-rCnum**, where **num** equals:

1 results in **OFFICIAL FILE COPY**.
2 results in **DATE FILE COPY**.
3 results in **DRAFT** with single-spacing and default paragraph style.
4 results in **DRAFT** with double-spacing and 10-space paragraph indent.

With **3** and **4**, the default page footer contains the date. See "Page Footer" in topic 2.7.4.3.2 for further information.

D Holds a value that can set debug mode. The flag **-D1** sets ***debug mode***; it requests the formatter to attempt to continue processing even if **mm** detects errors that would otherwise cause the processing to be canceled. It also causes **mm** to include a Source Code Control System (SCCS) release level information string, which is stored in the **RE** string register, in the default page header. See SCCS in the *AIX Operating System Technical Reference*.

E Holds a value that controls the font of the **Subject/Date/From** fields. A flag of the form **-rEnum** sets the font to be used. If **num** is **0**, these fields are in Roman font (or the "regular text" that is mounted on physical device 1 of the **troff** formatter). If **num** is **1**, the font is set to bold. The default value of number register **D** for **nroff** is **0**, and the default value for **troff** is **1**.

L Holds a value that sets the length of the physical page in lines to the value stored in register **L**. It is set by a flag of the form **-rLval**. The default **nroff** value is **66**, that is, 66 lines per page.

The default **troff** value is **11i**, eleven inches. You can set this parameter differently.

N Holds a value that specifies page numbering style; the flag form is **-rNnum**.

When **num** is **0** (the default), all pages get the prevailing header.
When **num** is **1**, the page header replaces the footer on page 1 only. Page 2 and subsequent pages get the prevailing header.
When **num** is **2**, the page header is omitted from page 1, and the subsequent pages get the prevailing header.
When **num** is **3**, **section-page** numbering occurs in the default page footer; the default page header does not print the page number. This affects both page and footnote reference numbering.
When **num** is 4, the default page header not printed although a user-specified header defined by the macro **.PH** is not affected.
When **num** is 5, section-page and section-figure numbering occurs.

The contents of the user-specified header and footer do not depend on the value of the number register **N**. **N** controls:

The default header and footer
Where the header or the footer is printed
The page, footnote, and figure numbering style.

If the prevailing header and footer are null, the value of **N** is not used, except for footnote and reference numbering and section-figure numbering with memorandum types 3 and 5.

O Stores a value that sets the amount of offset (to the right) for output. In **nroff** the value is unscaled, but in **troff** the value is scaled. The flag is of the form **-rOval** (O is the capital letter, not zero). You may find it useful for adjusting output positioning on some terminals. If this register is not set on the command line, the default offset is 0.75 inches (**-rO.75i**) in **nroff** and 0.5 inches (**-rO.5i**) in **troff**.

P Holds a value that specifies the starting page number of the document. The flag is of the form **-rPnum**. You can reset this register as often as you wish in the input text using an **.nr** request.

S Holds a value that establishes the point size and vertical spacing for the document for **troff**; **nroff** does not use this register. The flag is of the form **-rSnum**, where **num** is the type size in points. The default **num** is **10**, which specifies 10-point type on 12-point vertical spacing, giving 6 lines per inch (see "troff Point Size and Vertical Spacing" in topic 2.7.5.5). Vertical spacing is normally kept by **troff** at two points greater than the type size.

T Holds a value that **nroff** uses to derive some register settings for certain printers. **troff** does not use this register. The page length is stored in the formatter register **.p**, and the offset is stored in formatter register **.o**. The form of the flag is **-rTnum**.

The default value for **num** is **0**. The **nroff** default number of lines per page is set at 66, and page offset is set to zero.
If **num** is **1**, the number of lines per page is set to 80 and page offset is set to 3. This setting is intended for 12-character-per-inch printers.
If **num** is **2**, the page length is set to 84 lines per page and

underlining requests are ignored.
Other **num** values are not defined by the memorandum macros.

U Holds a value that **nroff** uses to control underlining of section headings. **troff** ignores this register. The form of the command line flag is **-rUnum**.

The default **num** is **0**, which causes a continuous underline. All heading characters including spaces are underlined.
If **num** is **1**, only letters and digits are underlined.

W Holds a value that is used to set page width (line and title length). The form of the command line flag is **-rWval**. The default **val** is 6 inches (**-rW6i** in **troff**), which in **nroff** is 60 characters line length in 10-pitch (**-rW60**), or 72 characters in 12-pitch (**-rW72**).

Note: The maximum setting that **troff** can accept is 7.54 inches (**-rW7.54i**).

2.7.2.4.6 Initialization Files

If you need a large number of parameters on the command line for some formatting application or if you use a particular setup frequently, you may find it convenient to set up the first (or only) input file of a document by entering:

1. Initialization of registers, if any, listed under "Setting Number Registers from the Command Line" in topic 2.7.2.4.5.

2. A **.so** (source) request:

   .so /usr/lib/umac/tmac.m.

3. The remainder of text.

In this case, you must not use the **-mm** flag with a formatter command, nor the **mm** or **mmt** commands; the **.so** request has the equivalent effect, but the registers discussed under "Setting Number Registers from the Command Line" in topic 2.7.2.4.5 must be initialized before the **.so** request. These registers values are meaningful *only* if they are set before the macro definitions are processed; it is generally preferable to set them using the **-r** flag.

When you use an initialization file, it is best to set within the 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 "INTRODUCTION"
.
.
.
```

specifies and sets registers for **nroff**, a line length of 80, a page offset of 10, and section-page numbering before loading the memorandum macros and subsequent text. You can use this same file initialization method to load other files of macros in macros in addition to (or instead of) the memorandum macros; this can include your own custom macro file.

2.7.2.5 Formatting Concepts

The normal action of the **troff** and **nroff** formatters is to ***fill*** output lines from one or more input lines; the formatter moves words up from subsequent lines and adds them to each output line until it cannot hold any more words before starting another output line. The output lines can be ***justified*** so that margins are aligned. Text is automatically ***left-justified***; that is, aligned at the left margin. Text can also be ***right-justified*** making it justified on both margins. In some special contexts, such as creating tables, text can be right-justified only (see Chapter 10, "Formatting Tables"). If text is completely justified, spaces are padded (see "Constant Spaces and Character Translation" in topic 2.7.2.5.3) between words (**nroff**) or between words and letters (**troff**) to make both margins align. As the formatter fills lines, it performs ***hyphenation*** of words as necessary to avoid excessive blank space within or between words.

You can turn modes on or off. Turning off fill mode also turns off justification and hyphenation.

Certain formatting requests and macros cause a ***break***; the filling of the current output line stops, the line (of whatever length) is printed, and the subsequent text begins a new output line. A few formatter requests and many of the memorandum macros cause a break.

While you can use formatter requests with the **mm** command, you should fully understand the consequences and side-effects that each request can have. Actually you will have little need to use formatter requests.

It is a good procedure to use formatter requests only when you must; instead use filter requests and the memorandum macros that are available when possible. This reduces your formatting workload and keeps text easier to alter.

You will find it easier to revise and edit your text if you make it your practice to keep input lines short and break them at the end of clauses and sentences. The formatter will fill lines for output.

Subtopics
2.7.2.5.1 Double Quote Delimiter
2.7.2.5.2 Null String
2.7.2.5.3 Constant Spaces and Character Translation
2.7.2.5.4 Hyphenation
2.7.2.5.5 Dashes, Minus Signs, and Hyphens
2.7.2.5.6 Tabs
2.7.2.5.7 Bullets
2.7.2.5.8 The Trademark
2.7.2.5.9 The BEL Character

2.7.2.5.1 Double Quote Delimiter

You must enclose any macro parameter containing ordinary blank spaces in double quote (") characters. Otherwise, the parameter is treated as several separate parameters. Double quotes are reserved as parameter delimiters in **mm**; you cannot use them as part of the value of a macro parameter or of a string that is to be used as a macro parameter. You can use two grave accents (` `) or two acute accents or apostrophes (' ') instead. This restriction is sometimes inconvenient, but it is necessary because many macro parameters are processed (interpreted) by the filter and by the formatter a variable number of times. For example, headings are first printed in the text and can be printed in a table of contents.

2.7.2.5.2 Null String

For any macro call, a ***null string*** is a special parameter with a printed width of zero. The preferred form for a null string is "". (There is no space between the double quote delimiters.) Such a parameter often has a special meaning.

Note: Omitting a parameter is not the same as supplying a null string. Furthermore, omitted parameters can only be used at the end of a parameter list; you can use a null string anywhere.

2.7.2.5.3 Constant Spaces and Character Translation

When you right-justify lines, the formatter can pad existing spaces in the line by appending additional spaces to them. This can change the appearance of text. You can avoid this problem by specifying a ***constant space***, which cannot be expanded during justification. There are two ways to accomplish this:

1. You can type a backslash followed by a space (\ ). This pair of characters directly generates a constant space.

2. You can sacrifice some seldom-used character to be ***translated*** into a space upon output.

Because the translation occurs after justification, you can use the chosen character anywhere you desire a constant space. The ~ (tilde) is often used for this purpose for English language text. To use the tilde as a substitute character, insert the following **.tr** (translation) request at the beginning of the document:

```
.tr ~
```

If a tilde must actually appear in the output, it can be temporarily treated literally by inserting:

```
.tr ~ ~
```

before the place where it is needed. Its previous usage is restored by repeating the **.tr ~** request, but only after a break or after the line containing the tilde has been forced out. Using the tilde in this fashion is **not** recommended for documents in which the tilde is often needed literally, as for some languages, or where they are needed within equations. Fortunately, there are few circumstances where all available characters are needed for literal output; there are some characters you can translate as a constant blank or translate for some other special purpose.

2.7.2.5.4 Hyphenation

The formatters do not perform hyphenation unless you request it. Hyphenation is performed in accordance with general (United States) English rules of hyphenation, but you are given methods to control the process. You can control whether and where hyphenation occurs in a character string.

You can turn hyphenation on for the body of the text by using the hyphenation request once at the beginning of the document:

```
.nr Hy 1
```

For hyphenation within footnotes and across pages, see "Format of Footnote Text" in topic 2.7.9.3.

If hyphenation is on, the formatters will automatically hyphenate words, if needed. However, you can specify the hyphenation points for a specific occurrence of any word by using a special character sequence known as a ***hyphenation indicator***, or by supplying a small list of words (about 128 characters total) with hyphenation points indicated so the formatter can check.

If you use the hyphenation indicator (the default sequence is **\%**) at the beginning of a word, the formatter does not hyphenate that word but keeps it intact by moving it to the next line. You can also use the hyphenation indicator to indicate legal hyphenation points inside a word. If you place the hyphenation indicator in a word, the formatter will, when hyphenation is needed, hyphenate **only** where a hyphenation indicator is inserted. If you wish to permit more than one possible hyphenation of a word, you must insert a hyphenation indicator in each place that hyphenation is permitted. In any case, **all** occurrences of the indicator disappear on output.

You can specify a different hyphenation indicator. For example, the circumflex ((**^**)) is often used for this purpose; this is done by inserting the following at the beginning of a document:

```
.HC (^)
```

Note: Any word containing explicit hyphens (as in fleur-de-lis) or dashes (also known as ***em dashes***, as in "Return to sender--Address unknown") is hyphenated immediately after a hyphen or dash if the formatter finds it necessary to hyphenate the word, even if the formatter hyphenation function is turned off. Use a hyphenation indicator at the front of the word to prevent breaking at the hyphen, or use the **.hy** request to place the word in a small list of words that cannot be hyphenated.

You can supply, via the formatter **.hw** request (see page 2.8.5.1), a small list of words with the permitted hyphenation points indicated. For example, to indicate the proper hyphenation of the word **printout**, you can specify:

```
.hw print-out
```

which will prevent the formatter causing the break **prin-tout**.

2.7.2.5.5 Dashes, Minus Signs, and Hyphens

The **troff** formatting processor has distinct graphics for a dash, a minus sign, and a hyphen, while **nroff** does not. Unfortunately, these characters cannot be represented in a way that is both compatible and convenient, but this is the most universally used convention for dash entry:

-- Dash Enter ***(EM** for each text dash for both **nroff** and **troff**. This string generates an em dash (**--**) in **troff** and two minus signs (**--**) in **nroff**.

- Hyphen Enter a minus sign (**-**) and use as is for both formatters. **nroff** prints a minus sign (**-**) and **troff** prints a true hyphen (**-**).

- Minus Enter **\-** for a true minus sign, regardless of formatter. **nroff** ignores the backslash while **troff** interprets the request for a literal minus sign.

Note: The **nroff** formatter interprets the minus, dash, and hyphen as the same character. In most fonts, this character is the width of a letter n. In typesetter terminology this is an ***en dash***, because it is an ***en-width*** character.

2.7.2.5.6 Tabs

The macros **.MT**, **.TC**, and **.CS** use the formatter **.ta** (tab) request to set tab stops and then restore the default. The default tab settings are every eight characters in **nroff** and every half inch in **troff**.

A tab character is always 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 mode (see "Formatting Concepts" in topic 2.7.2.5).

Note: The **tbl** filter changes tab stops but does not restore default tab settings.

2.7.2.5.7 Bullets

The **nroff** formatter often prints a bullet ( ) as a letter **o** overstruck by a plus sign (**+**). For compatibility with **troff**, the memorandum macros define a bullet string for use as appropriate by **troff** or **nroff**; the string is stored in string register **BU**. Rather than specifying overstriking, use the escape request sequence ***(BU** whenever a bullet is desired. **troff** interprets the request as a bullet character call.

The **.BL** (bullet list) macro uses this string to generate the bullets for the list items (see "Bullet List and Dash List" in topic 2.7.8.2.2).

2.7.2.5.8 The Trademark

A **Tm** (trademark) string is called with the ***(Tm** escape sequence. The string that is stored in string register **Tm** is interpreted as requests that place the letters **TM** one-half line above the text that it follows. For example:

The DisplayWrite \ *(Tm Self-Study Book is available at the public library.

yields (in **troff)**:

The DisplayWrite(TM) Self-Study Book is available at the public library.

Note: In **nroff**, the superscript (TM) is (except for some dot-matrix printers) the same size characters as the main text. Some printers do not have half-line scrolling capabilities.

2.7.2.5.9 The BEL Character

The nonprinting ASCII character BEL is used internally by many of the memorandum macros. You should avoid using the BEL character in your input; you could interfere with proper macro function and lose or scramble output.

2.7.2.6 Using Formatter Requests

Most formatter requests should not be used with memorandum macros because the **mm** command provides the corresponding formatting functions in a much more orderly and user-oriented fashion than do the basic formatter requests.

Note: If you use other formatter requests than those in the following list, you may inadvertently interfere with proper function of macros. This can cause scrambled output or lost output.

However, these formatter requests are useful and can be used with the **mm** command:

.af Assign Format (see topic 3.13.4).

.br Break (see topic 2.8.15.4).

.ce Center (see topic 2.8.15.4).

.de Define macro (see topic 2.8.8).

.ds Define String (see topic 2.8.8.1).

.fi Fill lines (see topic 2.8.15.4).

.fp Font Position (used for **troff** only; see topic 2.8.3.1).

.hw Hyphenate Word (see topic 2.7.2.5.4, 2.8.5.1).

.lg Ligature setting (**troff** only; see topic 2.8.16.3.2).

.ls Line Spacing (see topic 2.8.13.1).

.nf No Fill (see topic 2.8.15.4).

.nr Number Register (see topic 2.7.1, 2.8.9.1).

.nx Next file (see topic 2.8.18.1).

.rm Remove request, macro, or string register (see topic 2.8.8).

.rr Remove Register (see topic 2.8.9.1).

.rs Restore Spacing (see topic 2.8.13.4).

.so Source file set (see topic 2.8.18.1).

.sp Space vertical distance (see topic 2.8.15.4).

.ss Set space character size (**troff** only; see topic 2.8.5.2).

.ta Tab settings (see topic 2.7.2.5.6, 2.8.16.2.2).

.ti Temporary Indent (see topic 2.8.15.4).

.tl Title, three part (see topic 2.8.14.1).

.tr Translate (see topic 2.8.16.3).

.! Command (see topic 2.8.17).

2.7.3 Document Style

The memorandum macros let you conveniently select the overall style of a document. You do not need to provide detailed instructions for formatting cover sheets, tables of contents, memorandum headers, and so forth; these are automatically handled using the document style macros.

There are four basic types of documents you can prepare with memorandum macros by selecting a memorandum style, publication style, letter style, or general style. The first three types are selected with various parameters to the **.MT** memorandum type macro; the general style results when you **omit** the **.MT** call.

The two main document styles offered are ***memorandum style*** and ***publication style***. These styles, originally developed for research laboratory use, offer a standard and consistent method for preparation of memorandums and publications. They have special requirements for the first page and for the cover sheet. The memorandum and publication styles offer major ranges of options and differ sharply from each other. The information needed for both the memorandum and publication styles is much the same (title, author, date, and so on); a parameter to a single macro sets which style is used. There is only one selection parameter for publication style, but there are many choices for memorandum style. The differences between types of memorandums are almost entirely in the message that is printed, if any, which identifies the memorandum type.

Subtopics

2.7.3.1 Style Macros

These are style macros that are not used if the **.MT** macro is not called.

Macro	Description
.AE	Abstract end macro.
.AF	Alternate first page format macro suppresses or alters first page data for memorandum style documents.
.AS	Abstract start macro is used for memorandum and publication style documents.
.AT	Author title macro is used for memorandum and publication style documents.
.AU	Author macro is used for memorandum, publication, and letter style documents.
.AV	Approver macro is used for memorandum style documents.

Macro	Description
.CS	Cover sheet macro is used for memorandum and publication style documents.
.FC	Formal closing is used for letter style documents.
.MT	Memorandum type macro is used for memorandum, publication, and letter style documents.
.ND	New date macro sets the date used in a document.
.NE	Notations end macro.
.NS	Notations start macro is used for memorandum and letter style documents.
.OK	Other keywords macro is used for memorandum and publication style documents that use a cover sheet.
.SG	Typist data macro is used for letter and memorandum style documents.
.TL	Title macro sets the title information used in a memorandum or publication style document.

2.7.3.2 Style Macro Sequence

There are a number of style macros that are used for various styles; the following style ***opening macros*** must appear at the start of a document before text processing begins (except for the **.CS** macro). They are ignored if the **.MT** macro is not used. None of these macros are required, but if given they must appear in this order:

```
.ND new-date
.TL
one or more lines of text
.AF [company-name]
.AU name [initials] [loc] [dept] [ext] [room] [parm7] [parm8] [parm9]
.AT [title] ...
.AS [parm] [indent]
one or more lines of text
.AE
.OK [keyword] ...
.MT [type] [addressee]
.TC [slevel] [spacing] [tlevel] [tab] [hdg1] ... [hdg5]
.CS [pages] [other] [total] [figs] [tbls] [refs]
```

The only required macros for a memorandum or a released paper are **.TL** title macro, **.AU** author macro, and **.MT** memorandum type macro. You can omit the others (and their associated input lines) if you do not need the features they provide. Letter style and general style do not use **.TL**, **.MT**, **.AV**; letter style also does not use **.AF**, **.AS**, or **.AE**.

Note: Once you call **.MT**, you cannot again call any of the above listed macros (except the **.CS** macro), because they are removed from the table of defined macros to save space. The **.CS** macro, if used, must follow all other requests, macros, and text.

Additional style macros are ***closing macros*** for text features that may appear at the end of a document. You can place closing macros anywhere in a document, but they must precede the **.CS** macro. These are closing macros:

```
.NS [parm]
[optional line(s) of notation text]
.AV "approver name"
.FC [closing]
.SG [typist] [1]
.
.
.
[optional lines of author reference data (and requests).]
```

The **.SG** macro, if used, must be preceded by an **.FC** macro. A single **.NE** macro closes any number of preceding **.NS** macros. The **.AV** and **.CS** macros are ignored for letter style; only letter style uses the **.FC** macro. For publication style, the **.NS**, **.NE**, **.AV**, **.FC**, and **.SG** macros are also ignored.

2.7.3.3 Memorandum Types and Memorandum Type Field

The **.MT** (memorandum type) macro defines what type any given document is, even if it is not a memorandum. The **.MT** macro establishes the type of document and controls the format of the ***memorandum type field***, a text line that appears near the top part of the first page of a memorandum or of a publication style document. It also controls the format of cover sheets (see "Cover Sheet" in topic 2.7.3.11):

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

Legal values for **type** and the corresponding results are:

"" A null string results in no memorandum type header. A simple letter format results when **""** or **"0"** is the **type** value.

0 A zero gives the same results as a null string.

string A **string** longer than one character results in a memorandum type header that is the string. **mm** tries to interpret a one-character string as a legal **type**.

none No parameter results in a memorandum header of **MEMORANDUM FOR FILE** (the default value, **1**).

1 A value of **1** results in a memorandum header of **MEMORANDUM FOR FILE**.

2 A value of **2** results in a memorandum header of **PROGRAMMER'S NOTES**.

3 A value of **3** results in a memorandum header of **ENGINEER'S NOTES**.

4 A value of **4** causes a publication style, which uses no memorandum type field.

5 A value of **5** causes a letter style, which uses no memorandum type field.

If **type** indicates a labeled memorandum style (the value of **type** is not **""**, **0**, **4**, or **5**) and is longer than one character, then the memorandum type header is printed in a field following the author information block. For example:

```
.MT "Technical Note #5"
```

causes **Technical Note #5** to print on the line immediately following the last line of author information.

The **parm2** parameter to **.MT** is the name of the addressee of a letter. If present, that name and the page number replace the memorandum page header (see "Page Layout and Pagination" in topic 2.7.4) on the second and following pages of a letter:

```
.MT 1 "John Jones"
```

produces the following header on page 2:

```
John Jones -- 2
```

Note: This parameter is not used for this purpose if the first parameter is **4** (for the publication style). In that case, **parm2** is a number

1, changing the presentation format of a release paper document to support multiple authorship.

Subtopics
2.7.3.3.1 Publication Style
2.7.3.3.2 General Style
2.7.3.3.3 Letter Style

2.7.3.3.1 Publication Style

The publication style (**.MT 4 [1]**) results in a centered, bold title followed by centered names of authors. The location (from the **.AU loc** parameter, see "Author(s)" in topic 2.7.3.7) of the last author is used as the location of origin of the paper, following the company name. If you give the optional **1** second parameter to **.MT 4**, the name of each author is followed by the respective company name and location.

Information necessary for the memorandum style but not for the publication style is ignored. For example, the closing macros described under "Macros for the End of a Memorandum" and their associated lines of input are ignored when you specify publication style.

You can list authors and their data in the publication style by setting an **.AF** macro appropriately and defining a string (with a two-character string register such as XX) for the address before each **.AU**. For example:

```
.TL
A Learned Treatise
.AF "Nuwait Control Systems, Inc."
.ds XX "222 Avenida de Armadillo, Port Lee 09999"
.AU "B. Sitter"  " " XX
.AF "Western Swing and Gate Ltd."
.ds ZZ "Ste. 644, Vista del Camino Building, Hounds Bay 09990"
.AU "Sherlock Q. Watson"  " " ZZ
.MT 4 1
```

2.7.3.3.2 General Style

The general document style occurs when you omit the **.MT** macro. You get no cover letter, table of contents, or abstract. You should avoid using the style macros when you elect to use this style; many of them cause processing errors if not omitted.

2.7.3.3.3 Letter Style

In the letter style (**.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 assumed that preprinted stationery is used, providing a company logo and address. No cover page is generated, and no abstract is printed.

2.7.3.4 Setting the Date

By default, the current date appears in the ***date field*** of a memorandum. You can set any date you want in this field using the **.ND** (new date) macro:

```
.ND new-date
```

The **.ND** macro alters the string held in the string register **DT**, which initially holds the current date.

2.7.3.5 Title

The title of the memorandum or paper follows the **.TL** (title) macro and is processed in fill mode. For example:

```
.TL
Hickory, Dickory, Dock
```

You can break the title into several lines on output, using the **.br** request as follows:

```
.TL
Hickory
.br
\! .br
Dickory
.br
\! .br
Dock
```

On output, the title appears after the word **subject** in the memorandum style. In the publication style, the title is centered and in boldface type.

2.7.3.6 Alternate First-Page Format

You can delete the words **subject**, **date**, and **from** (in a memorandum style) and give an alternate company name by using the **.AF** (alternate format) macro:

```
.AF [company-name]
```

If a parameter is given, it replaces the initial company name without affecting the other headings; it sets the value in string register **In** (see "Bottom-of-Page Processing," 2.7.4.9). If the parameter is a null string, the default company name is suppressed; in this case, extra blank lines are inserted in the output to allow room for a logo.

If you are using **troff**, the only appropriate use for **.AF** is to specify a replacement for the default company name.

.AF with no parameter suppresses the default company name and the **Subject/Date/From headings**, thus allowing output on preprinted stationery. The use of **.AF** with no parameters is equivalent to the use of the **-rA1** flag on the **mm** command line (see "Setting Number Registers from the Command Line" in topic 2.7.2.4.5), except that the latter is necessary if you wish to change the line length or page offset (which default to **5.8i** and **1i** respectively, for preprinted forms).

Note: The command line flags **-rO** and **-rW** are not effective if the **.AF** macro is used.

The command line flag **-rEnum** controls the font of the **Subject/Date/From** block (see "Setting Number Registers from the Command Line" in topic 2.7.2.4.5).

2.7.3.7 Author(s)

The **.AU** (author) macro identifies an author of a document. This macro has the form:

```
.AU name [initials] [loc] [dept] [ext] [room] [parm7] [parm8] [parm9]
.AT [title] ...
```

You must give a **name** parameter, and can give up to nine parameters. You can call the author macro as often as necessary to identify all the authors of a document.

You must give the first six parameters in the order shown. If any parameter contains blanks, you must enclose that parameter within double quotes. An **.AT** (author title) macro can follow each **.AU** macro on the next input line. The parameters are used as follows:

name In the memorandum style (see "Memorandum Types and Memorandum Type Field" in topic 2.7.3.3), each author's name is output in the ***from field***, followed by location and department number on one line and by room number and extension number on the next; each author's name will appear in the ***signature block*** on a separate line following space for the signature (see "Macros for the End of a Memorandum" in topic 2.7.3.10).

loc The **loc**, if present, must be a two-character user-defined string register name that is defined as explained under "Publication Style" in topic 2.7.3.3.1 to provide the author's address.

dept The **dept** parameter is a character string.

ext The **ext** is a four-digit numeric string. The **x** prefix for the telephone extension is added to output automatically.

room The **room** parameter is a string of numbers and characters.

parm(s) Undefined parameters 7 through 9 of the **.AU** macro, if present, follow this author information on the ***from field***, each on a separate line. You can use these last three parameters for organizational numbering schemes, etc. For example:

```
.AU "S. P. Lename" SPL IH 9988 7766 5H-444 "" 070750LK
```

On output, the room number **5H-444** would be followed by a blank line because of a null string in the seventh parameter; **070750LK** would follow on the next line.

The printing of the location, department number, extension number, and room number are suppressed on the first page of a memorandum if you set the **Au** register to **0**; the default value for **Au** is **1**.

The name, initials, location, and department are also used in the signature block (see "Signature Block" in topic 2.7.3.10.1). The author information in the from field as well as the names and initials in the signature block will appear in the same order as the **.AU** macros.

The names of the authors in the publication style are centered below the title. After the name of the last author, the name of the company and the location are centered.

You can use the **.AT** (author title) macro to specify certain information about an author of a document. The **.AT**, if present, must immediately follow the **.AU** for the given author. For example:

```
.AU "J. Joli Jones" JJJ MI 9876 5432 1Z-123
.AT "Administrator, Product Testing Laboratory"
.AU "Sydney S. Smith" SSS TX 9876 5678 B-23
.AT "Consumer Support Services"
```

If you have blanks in a parameter, use double-quote delimiters as shown above. Author title fields, if used, follow the signature block for each author.

2.7.3.8 Abstract and Memorandum for File Cover Sheet

The placement of an abstract varies with the document style you select (see "Memorandum Types and Memorandum Type Field" in topic 2.7.3.3). It is placed on the cover sheet (see "Cover Sheet" in topic 2.7.3.11) or begins on the first page of the document, or both, depending on the selected document style.

There are two styles of cover sheets: publication and memorandum. The memorandum style of cover sheet is used for all document styles except for publication style and letter style. There are minor variations in memorandum cover sheet appearance depending on memorandum type selections.

Note: You cannot get an abstract or a cover sheet if you select letter style.

You can call the optional abstract with the **.AS** (abstract start) macro and close it with the **.AE** (abstract end) macro:

```
.AS [cover] [indent]
text of the abstract
.AE
```

If the **cover** parameter of **.AS** is **0** and:

- The selected style is a publication or a memorandum, the abstract is printed on page 1 and on the cover sheet.
- The selected style is not publication, memorandum, or letter, the abstract begins on page 1 and no cover sheet is printed.

If the **cover** parameter of **.AS** is **1** and:

- The style is memorandum, the abstract appears only on the cover sheet (if any).
- The style is not memorandum or letter, the abstract is printed on page 1.

If the first parameter of **.AS** is **2** and the document style is not **letter**, the abstract appears only on the cover sheet(s). In this case only, the cover sheets are produced automatically by **.AS**, without invoking the **.CS** macro.

If the **indent** parameter is specified as a number, the abstract is indented this number of en spaces from each margin. Indentation values must be unscaled. The default is for the abstract to print with ordinary text margins.

Notations such as a **copy to** list (see "Closing Notations" in topic 2.7.3.10.1) are allowed on memorandum cover sheets; the **.NS** and **.NE** macros must appear after the **.AS 2** and **.AE**. Headings and displays are not permitted within an abstract.

2.7.3.9 Other Keys

An **.OK** (other keys) macro lets you store topics and keys pertaining to a memorandum for a technical memorandum cover sheet. You can specify up to nine such keys or key phrases as string parameters to the **.OK** macro; if any key string contains spaces, you must enclose it within double quotes:

```
.OK [keyword] ["key words"] ...
```

2.7.3.10 Macros for the End of a Memorandum

You can request the signatures of the authors and a list of notations at the end of a memorandum (but not of a publication). The following macros and their input are ignored if the publication style is selected.

Subtopics
2.7.3.10.1 Signature Block

2.7.3.10.1 Signature Block

The **.FC** (formal closing) macro prints **Yours very truly** as a formal closing. You must give this macro before the **.SG** (signature) macro, which prints the signer's name:

```
.FC [closing]
.SG [typist] [1]
```

You can specify a different formal closing as a parameter to **.FC**. This macro is ignored for publication style (see "Memorandum Types and Memorandum Type Field" in topic 2.7.3.3).

.SG prints the author 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 you give no parameters, the line of reference data is not printed. (***Reference Data*** is the following: location code, department number, author's initials, and typist's initials, all separated by hyphens at output.) A non-null first parameter is treated as the typist's initials and is appended to the reference data. Supply a null first parameter to print the reference data without the typist's initials or the preceding hyphen.

If there are several authors and if the second parameter is given, then the reference data is placed on the same line as the name of the first, rather than last, author.

The reference data contains only the location and department number of the first given author. Thus, if there are authors from different departments and/or from different locations, you should supply the reference data manually after the call (without parameters) of the **.SG** macro. For example:

```
.SG
.rs
.sp -lv
PY/MH-9876/5432-JJJ/SPL-cen
```

Closing Notations

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

After the signature and reference data, many types of notations can follow, such as a list of attachments or *copy to* lists. The various notations are obtained through the **.NS** (notation specification) macro, which provides for the proper spacing and for breaking the notations across pages, if necessary.

The codes for **parm** and the corresponding notations are:

string	Copy (**string**) to
(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
```

Note: Codes 8 and 9 are often used as opening notations.

If **parm** consists of more than one character, it is placed within parentheses between the words **Copy** and **to**. For example:

```
  .NS "with att. 1 only"
```

This generates:

```
Copy (with att. 1 only) to
```

as the notation.

You can specify more than one notation before the **.NE**, because an **.NS** macro terminates the preceding notation, if any; an **.NE** closes the last preceding **.NS** macro. For example:

```
  .NS 4
  Attachment 1-List of register names
  Attachment 2-List of string register and macro names
  .NS 1
  J. J. Jones
  .NS 2
  S. P. Lename
  .NE
```

is formatted as:

```
Atts.
Attachment 1-List of register names
Attachment 2-List of string register and macro names

Copy (with att.) to
J. J. Jones

Copy (without att.) to
S. P. Lename
```

To place the notation list on the Memorandum for File cover sheet, use **.NS** and **.NE** macros following **.AS 2** and **.AE** (see "Abstract and Memorandum for File Cover Sheet" in topic 2.7.3.8).

Approval Signature Line

```
  .AV approver-name
```

You can use the **.AV** (approver) macro after the last notation block to automatically generate a line with spaces for the approval signature and date. For example:

```
  .AV "Ishmael"
```

produces:

```
APPROVED:


-----------------  ------
Ishmael            Date
```

The **.AV** macro is ignored in publication style (see "Publication Style" in topic 2.7.3.3.1).

2.7.3.11 Cover Sheet

The **.CS** macro generates a distinct cover sheet in either the publication or memorandum style.

.cs [**pages**] [**other**] [**total**] [**figs**] [**tbls**] [**refs**]

The parameters form a ***count data block***. All the other information for the cover sheet is obtained from data given before the **.MT** macro call (see "Style Macro Sequence" in topic 2.7.3.2). The count data is generated by **mm** automatically, but you may specifically alter the count by using parameter specifications. This is useful primarily if you have inserts or attachments with your document.

pages The pages of text.

other Other pages. The initial value is **0**.

total The sum of all pages.

figs The number of figure pages.

tbls The number of table pages.

refs The number of reference pages.

The cover sheet data of a memorandum includes a count data block and a key data block.

The count data that appears in the lower left corner of the technica memorandum cover sheet is the data shown in the preceding list.

The key data are printed if the **.OK** (see "Other Keys" in topic 2.7.3.9) macro is used in the beginning macros (see "Style Macro Sequence" in topic 2.7.3.2). This is unaffected by the presence or absence of the **.CS** parameters; but the key data, if given, are not used if the **.CS** macro is not called.

If the publication style is used, all the parameters to **.CS** (if given) are ignored. The **.CS** macro is placed at the end of a document.

Note: This section refers to cover sheets for memorandum and publication papers only; the mechanism for producing a memorandum for file cover sheet is discussed under "Abstract and Memorandum for File Cover Sheet" in topic 2.7.3.8. There is no cover sheet for letter style.

2.7.4 Page Layout and Pagination

Pages are laid out into several parts or zones. These zones include the body of the text, margins consisting of (generally) blank space separating text from the page edge and binding, and header and footer areas, which contain organizational information.

The terms used in this guide to describe page layout conform primarily to the mnemonic characters that comprise the corresponding macro (as in **even header** for the macro **.EH**). If you want to pursue formatting and typesetting seriously, be advised that terminology used here is not necessarily that of the typesetting profession. A typesetter is likely to call an **odd-page** a ***recto*** and an **even-page** a ***verso***; we *have* used that terminology in organizing "Page Layout and Pagination" information.

Subtopics

2.7.4.1 Top and Bottom Margin

The **.VM** (vertical margin) macro lets you specify extra space at the top and bottom of the page. This space precedes the page header and follows the page footer.

```
.VM [top] [bottom]
```

.VM takes two unscaled parameters and adds their values to the default values to determine the amount of vertical space at the top and bottom of the page, preceding the header and following the footer, for example:

```
.VM 10 15
```

In **nroff**, this adds 10 blank lines to the default top of the page margin and 15 blank lines to the default bottom of the page margin. Both parameters must be positive values. (You can decrease default spacing at the top of the page by defining **.TP** as in "Top-of-Page Processing" in topic 2.7.4.7.)

2.7.4.2 Right Margin Justification

You can use the **.SA** (set adjustment) macro to set right-margin justification for the main body of text.

.SA [**parm**]

The parameter of **.SA** is optional, but the results of leaving it off depend on context. Two justification flags are used, a current justification flag and default justification flag.

.SA 0 sets both flags to no justification; it acts like the **.na** formatter no-adjust request (see topic 2.8.5.2).

.SA 1 sets both flags to cause justification; it acts like the **.ad** adjust request (see topic 2.8.5.2).

.SA without a parameter causes the **current** flag to be copied from the **default** flag, thus performing either a **.na** or **.ad**, depending on what the default is. Initially, both flags are set for no justification in **nroff** and for justification in **troff**.

In general, you can use the formatter request **.na** to ensure that justification is turned off, and you should use the **.SA** macro to restore justification, rather than the **.ad** request. In this way, justification (or lack thereof) for the general body of the text is specified by inserting **.SA 0** or **.SA 1** only once at the beginning of the document.

2.7.4.3 Headers and Footers

Text that occurs at the top of each page is known as the ***page header***. Text printed at the bottom of each page is called the ***page footer***. There can be no more than three lines of text associated with the page header. There can be a line for:

    Every pag
    Even page onl
    Odd page only

Thus the page header of a page can have up to two lines of text

    The line that occurs at the top of every pag
    The line for the even- or odd-numbered page

The page footer has the same limitations as the page header.

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. An exception is **section-page numbering**, as specified under "Header and Footer with Section-Page Numbering" in topic 2.7.4.3.3. There are also exceptions for the first page. (See "First-Page Header" in topic 2.7.4.4.4 and "First-Page Footer" in topic 2.7.4.4.5.)

There are header macros to specify even and odd headers, and general, even, and odd footers. The macros are all called in the same way, and are of the form:

  .PF [**parm**]

The **parm** is a three-part parameter of the form:

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

On output, the parts are left-justified, centered, and right-justified, respectively. If it is inconvenient to use the apostrophe (') as the parameter delimiter, you can replace it (at every use in the parameter) by any other character; the character cannot be used anywhere within the parameter except as a delimiter.

You can place string register and number register names in the parameters to the header and footer macros. If you want the value of a register computed (or the string in the string register to be used) when the respective header or footer is printed, the call must be protected by four preceding backslashes. The string or register invocation is processed three times:

    As the parameter to the header or footer macr
    In a formatting request within the header or footer macr
    In a formatter **.tl** title request during header or footer processing.

Finally, it is read as a string called by an escape request or its value is taken (for a number register). At each stage, a backslash (or **escape**) is removed.

For example, you must preface the **P** (page number) register with four backslashes in order to specify a header in which the page number is printed at the right margin:

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

This creates a right-justified header (left and center portion of the title line are left blank) containing the word **Page** followed by the page number, arrived at by reading the value in the **P** register. Similarly, to specify a footer with the section-page style, you specify:

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

As another example, suppose you have defined a string register **a]** to contain the current section that is to be printed at the bottom of each page. The **.PF** (page footer) macro call then is:

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

If only one or two backslashes are used, the footer will print a constant value for **a]** at the place where the **.PF** appears in the input text. The value printed is its value when the last backslash is stripped away.

Subtopics
2.7.4.3.1 Page Header
2.7.4.3.2 Page Footer
2.7.4.3.3 Header and Footer with Section-Page Numbering

2.7.4.3.1 Page Header

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

.PH ["'**left-part'center-part'right-part**'"]

The initial value is:

"'-'\\\\P'-'"

which gives a centered page number enclosed by hyphens. The page number which is contained in the **P** register is by default an Arabic number. You can change the format of the number with the formatter request **.af** (see topic 3.13.4).

If debug mode is set using the **-rD1** flag on the command line, the SCCS data stored in the string register **RE** is included in the default header and printed at the top left of each page (see "Setting Number Registers from the Command Line" in topic 2.7.2.4.5). This consists of the SCCS **Release and Level of MM** running header, identifying the current memorandum macro release version followed by the current line number within the current input file. The **RE** string register holds the string of the SCCS release and revision level of the current version of **mm**; it can also be called any place in text using the **RE *release identification*** escape request. For example, entering:

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

produces (using **MM** v3.5):

```
This is version 3.5 of the macros.
```

2.7.4.3.2 Page Footer

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

.PF ["'**left-part'center-part'right-part**'"]

Its initial value yields a blank line. If you specify the **-rCnum** flag on the **mm** command line (see "Setting Number Registers from the Command Line" in topic 2.7.2.4.5), the type of **copy** follows the footer on a separate line. In particular, if you specify **-rC3** or **-rC4** (**DRAFT**), the footer is also initialized to contain the date instead of a blank line (see "Setting the Date" in topic 2.7.3.4).

2.7.4.3.3 Header and Footer with Section-Page Numbering

You can have pages numbered sequentially within sections. To get this numbering style, specify **-rN3** or **-rN5** on the **mm** command line. With ***section-page style***, the default footer is a centered ***section-page number***, with pages numbered with the section number and the page number within the section. The default header is blank. For instance, the second page of section 7 would have the page number **7-2** as a footer.

2.7.4.4 Recto Page

A ***recto page*** is an odd-numbered page of a text. It appears to the reader as a right-hand (recto) page when a book is opened. The memorandum macros handle odd-numbered pages in a particular way and treat the first page as a special case of a recto page.

Subtopics
2.7.4.4.1 Odd-Page Force
2.7.4.4.2 Odd-Page Header
2.7.4.4.3 Odd-Page Footer
2.7.4.4.4 First-Page Header
2.7.4.4.5 First-Page Footer

2.7.4.4.1 Odd-Page Force

The **.OP** (odd page) macro causes a break, and the text following it begins at the top of an odd-numbered page.

```
.OP
```

The results of an odd page force are:

If currently at the top of an odd page, no action takes plac

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

If currently on an odd page (but not at the top of the page), on blank (even) page is produced, and printing resumes on the following odd page.

2.7.4.4.2 Odd-Page Header

The **.OH** (odd page header) macro supplies a line that is printed at the top of each odd-numbered page, immediately following the header. The initial value gives a blank line.

```
.OH ["'left-part'center-part'right-part'"]
```

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

2.7.4.4.3 Odd-Page Footer

The **.OF** (odd page footer) macro supplies a line that is printed at the bottom of each odd-numbered page, immediately above the footer line.

```
.OF ["'left-part'center-part'right-part'"]
```

The default value results in a blank line.

2.7.4.4.4 First-Page Header

In a memorandum or a publication style document, the page header on the first page is automatically suppressed if a break does not occur before **.MT** is called. The macros and text of beginning macros, header and footer macros, and the **.nr** and **.ds** requests do not cause a break and you can use them before the **.MT** macro call (see "Memorandum Types and Memorandum Type Field" in topic 2.7.3.3). If you use a macro or request and want to be sure that it does not cause a break, preface it with an apostrophe instead of a period.

2.7.4.4.5 First-Page Footer

By default, the first page footer is a blank line. If, in the input text, you specify **.PF** or **.OF** before the end of the first page of the document, their footer lines will appear at the bottom of the first page.

Note: The header (whatever its contents) replaces the footer **on the first page only** if you specify the **-rN1** flag on the command line (see "Setting Number Registers from the Command Line" in topic 2.7.2.4.5).

2.7.4.5 Verso Page

A ***verso page*** is an even-numbered page; it appears on the left when viewing an open book. It is the reverse (verso) side of an (odd-numbered) page.

Subtopics
2.7.4.5.1 Even-Page Header
2.7.4.5.2 Even-Page Footer

2.7.4.5.1 Even-Page Header

The **.EH** (even page header) macro supplies a line that is printed at the top of each even-numbered page, immediately following the header line. The initial value gives a blank line.

```
.EH ["'left-part'center-part'right-part'"]
```

This macro is the same as **.OH**, except that it applies to even-numbered pages.

2.7.4.5.2 Even-Page Footer

The **.EF** (even page footer) macro supplies a line that is printed at the bottom of each even-numbered page, immediately **preceding** the footer.

```
.EF ["'left-part'center-part'right-part'"]
```

This macro is the same as **.OF**, except that it applies to even-numbered pages. The initial value is a blank line.

2.7.4.6 A Header and Footer Example

The following sequence specifies blank lines for the header and footer lines, page numbers on the outside edge of each page (that is, the top left margin of even pages and top right margin of odd pages), and **Revision 3** on the top inside margin of each page:

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

2.7.4.7 Top-of-Page Processing

There are two user-definable macros that **mm** uses during header processing. The **.TP** macro is called in the environment of the header; it performs top-of-page processing. The **.PX** macro, is a user-exit macro that is called when the normal environment is restored with **no-space mode** already in effect. A user experienced in writing macros can tailor these macros for customized top-of-page processing.

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 register **}t** contains the header string defined by the **.PH** macro.

The string register **}e** contains the even-page header string defined by the **.EH** macro.

The string register **}o** contains the odd-page header string defined by the **.OH** macro.

You can redefine the **.TP** macro to cause any desired header processing for more specialized page titles. (An example for two-column output can be found under "Two-Column Headings" in topic 2.7.4.8.2.)

Formatting done within the **.TP** macro is processed in a different environment from that of the body. You can obtain a page header that includes, for example, three centered lines of data -- a document's number, issue date, and revision date -- by defining **.TP** as follows:

```
.de TP
.sp
.ce 3
777-888-999
Iss. 2, AUG 1988
Rev. 7, SEP 1988
.sp
..
```

Since **.PX** lets you perform some operation as you exit from the header environment (***user-exit*** macro), you can use it to insert a text line in the normal page environment at the top of each page. The text can have, for instance, tab stops to align it with columns of text in the body of the document.

You can have the message **PRIVATE** centered and underlined on the second line of a document, preceding the page header, by setting the **Pv** (private) register:

```
.nr Pv num
```

The results of setting **Pv** register are:

If **num** is 0, **PRIVATE** is not printed. This is the default value.

If **num** is 1, **PRIVATE** prints on the first page only.

If **num** is 2, **PRIVATE** prints on all pages. This value causes an internally defined memorandum macro to print **PRIVATE** on the first page and causes **.TP** to be used. The default definition of the **.TP** macro prints **PRIVATE** on subsequent pages. If you want to define **.TP** for other actions, be sure to include instructions to print **PRIVATE**, or use the **.PM** macro instead (see "Bottom-of-Page Processing" in topic 2.7.4.9). Your definition of the **.TP** macro replaces the initial definition.

2.7.4.8 Two-Column Output

There are macros provided to give you two-column output, driving either **nroff** or **troff**. If **nroff** is used for printing two-column format, you will have to prepare for it on the command line with your terminal specification (see "Printing Two-Column Output" in topic 2.7.2.4.4).

Subtopics
2.7.4.8.1 Two-Column Format
2.7.4.8.2 Two-Column Headings

2.7.4.8.1 Two-Column Format

You can print two columns on a page with the **.2C** (two column) macro. Text following the **.2C** macro is formatted in two columns until a subsequent **.1C** (one column) macro restores single column format:

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

In two-column processing, each physical page is thought of as containing two columnar pages of equal (but smaller) page width, but page headers and footers are not affected.

Note: The **.2C** macro does not balance two-column output. If there is not enough text to fill a two-column page, the right column (if any) will be shorter than the left, which is filled before the right column begins.

You can have full page-width footnotes and displays when in two-column mode, although the default action is for footnotes and displays to be narrow in two-column mode and wide in one-column mode. Footnote and display width is controlled by a **.WC** (width control) macro:

```
.WC [flags]...
```

These are **.WC** parameters (flags):

Table 7-1. Width Control Macro Parameters

Flag	Meaning
N	***Normal*** is the default setting: **-WF -FF -WD FB**.
WF	***Wide footnote*** gives page-wide footnotes even in two-column mode.
-WF	***No wide footnote*** gives footnotes in column mode, wide in **1C** mode, narrow in **2C** mode, (unless **FF** is set).
FF	***First footnote*** gives all footnotes the same width as the first footnote encountered for that page.
-FF	***No first footnote*** gives footnote style following the settings of **WF** or **-WF**.

WD	***Wide displays*** causes wide displays always (even in two-column mode).
-WD	***No wide displays*** gives displays using the column mode in effect when the display is encountered.
FB	***Float break*** makes floating displays cause a break when output on the current page.
-FB	***No float break*** causes floating displays on current page to not break.

For example: **.WC WD FF** causes all displays to be wide, and all footnotes on a page to be the same width, while **.WC N** reinstates the default actions. If you give conflicting parameters to **.WC**, the last one given is used. For example, **.WC WF -WF** has the same effect as **.WC -WF**.

2.7.4.8.2 Two-Column Headings

In two-column output, it is sometimes necessary to have headers over each column, as well as headers over the entire page. You can cause this effect by redefining the **.TP** macro to give header lines both for the entire page and for each of the columns. For example:

```
.de TP
.sp 2
.tl 'Page \\nP'OVERALL''
.tl ''Title''
.sp
.nf
.ta 16C 31R 34 50C 65R
left&DIAMOND.center&DIAMOND.right&DIAMOND.left&DIAMOND.center&DIAMOND.right&
&DIAMOND.first column&DIAMOND.&DIAMOND.&DIAMOND.second column
.fi
.sp 2
..
```

The preceding example produces two lines of page header text plus two lines of headers over each column. (The &DIAMOND. symbol represents the tab character.) The tab stops are for a 65-en overall line length.

2.7.4.9 Bottom-of-Page Processing

You can specify lines of text to print at the bottom of each page following the footnotes (if any) but before the page footer. The text is enclosed between the **.BS** (bottom-block start) and **.BE** (bottom-block end) macros:

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

Note: The bottom block appears on the table of contents for Memorandum for File, but not on the Technical Memorandum or publication cover sheets.

You can remove this block of text by specifying an empty block:

```
.BS
.BE
```

The **.PM** (proprietary markings) macro attaches a **PRIVATE**, **NOTICE**, **PROPRIETARY**, or **RESTRICTED** notice to the page footer.

```
.PM [parm]
```

The parameter can be:

Table 7-2. Proprietary Markings Parameters

Parameter	Meaning
""	Turn off previous notice, if any.
none	Turn off previous notice, if any.
P	PRIVATE (See also "Top-of-Page Processing," in topic 2.7.4.7)
N	NOTICE
BP	<<Company Name>> PROPRIETARY
BR	<<Company Name>> RESTRICTED

If an unrecognized parameter is given for the **.PM** macro, a **NOTICE** message is given.

The **In** string register contains by default the string "<<Company Name>>" that the **.PM** macro uses. You can define this string to modify the form of the proprietary markings.

Note: You can use either the **.BS/.BE** or the **.PM** macro but not both in the same document.

2.7.4.10 Skipping Pages

The **.SK** (page skip) macro skips pages, but retains the usual header and footer processing.

.SK [**num**]

If **num** is omitted, null, or **0**, **.SK** skips to the top of the next page unless it is currently at the top of a page. The **num** parameter, if given, must be a positive whole number (or null). The parameter given is the number of pages that is skipped. That is, **.SK** always positions the text that follows it at the top of the following page. **.SK 1** always leaves one page that is completely blank except for the header and footer (unless it immediately follows an **.MT** macro); **.SK 2** always leaves two full blank pages, and so on.

2.7.4.11 Displays

Displays are blocks of text you wish to keep together--not split across pages. The formatter processes displays in an environment that is different from that of the body of the text. The memorandum macros offer two styles of displays:

Static display style initiated using the **.DS** macro.

Floating display style initiated using a **.DF** macro.

In the static style, the display appears in the same relative position in the output text as it does in the input text; this can result in extra white space at the bottom of a page if the display is too big to fit on a page following the text there.

In the floating style, the display **floats** through the input text to the top of the next page if there is not enough room for it on the current page; thus the input text that follows a floating display may actually precede it in the output text, preventing wasted blank space on the page preceding the display. **mm** queues floating displays so that their order of output is kept.

By default, **mm** processes a display in no-fill mode, single-spaced, and not indented from the existing margins. You can specify indentation or centering as well as fill-mode processing.

A display is output on the current page if there is enough room to contain the entire display on the page, or if the display is longer than one page in length and less than half of the current page is used. A full page width display (a ***wide display***) never fits on the second column of a two-column document.

Note: You cannot nest displays and footnotes in any combination whatsoever. Although lists and paragraphs are permitted, you cannot use **.H** or **.HU** headings within displays or footnotes.

Subtopics

2.7.4.11.1 Static Displays

A static display is started by the **.DS** (static display) macro and terminated by the **.DE** (display end) macro. With no parameters, **.DS** accepts the lines of text exactly as they are typed (no-fill mode) and does not indent them from the prevailing left margin indentation or from the right margin. Omitted parameters are assumed to be zero:

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

The **rindent** parameter is the indentation from the right margin that in **nroff** is the number of characters that the line length is decreased and in **troff** is a scaled length value. The **format** parameter to **.DS** 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	Standard indent
2 or C	Center each line
3 or CB	Center as a block.

The display format value **3** (**CB**) centers the entire display as a block, as compared to **.DS 2** and **.DF 2**, which center each line individually. That is, all the collected lines are left-justified, and then the display is centered based on the width of the longest line.

Note: This block centering format must be used in order for the equation formatter **mark** and **lineup** features to work with a centered equation (see "Equations" in topic 2.7.4.11.4).

The standard indentation is stored in the **Si** number register with an initial value of **5**. The text of an indented display aligns (by default) with the first line of indented paragraphs. The indent value is contained in the **Pi** number register (see "Block and Indented Paragraphs" in topic 2.7.7.1). Though their initial values are the same, these two registers are independent of each other.

The **fill** parameter can be given as an integer or letter value. A value given as **0**, **N**, or **""** (null value) results in no fill; a value of **1** or **F** results in text fill mode.

By default, **mm** places a half vertical space (one blank line, in **nroff**) before and after static and floating displays. You can inhibit vertical space separation for static displays by setting the **Ds** (static display) register to **0**.

The following example shows the usage of all three parameters for displays. This block of text is filled and indented 5 spaces from both the left and the right margins; therefore, it remains aligned (centered) with the other text.

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

```
  States of America."
  .DE
```

This results in:

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

2.7.4.11.2 Floating Displays

You can start a floating display with the **.DF** (floating display) macro and close it with the **.DE** macro.

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

The parameters have the same meanings as for **.DS** (see "Static Displays" in topic 2.7.4.11.1), except that with floating displays, indent, no indent, and centering are always calculated with respect to the initial left margin. (With floating displays the prevailing indent can change between the time when the formatter first reads the floating display and the time that the display is printed.) A half vertical space always occurs both before and after a floating display; there is no floating display equivalent to the **Ds** register.

You have great control over the output positioning of floating displays through the use of two number registers, **De** and **Df**. When the formatter encounters a floating display, it is processed and placed onto a queue of displays waiting to print. Displays are always removed from the queue and printed in the order that they appear in the input file. If a new floating display is encountered and the queue of displays is empty, then the new display is a candidate for immediate output on the current page. Immediate output is governed by the size of the display and the setting of the **Df** (display format) register. The **De** (display eject) register controls whether or not text appears on the current page after a floating display is produced.

As long as the queue contains one or more displays, additional displays are automatically stored rather than being immediately output. When a new page is started (or the top of the second column is started when in two-column mode), the next display from the queue is a candidate for output if the **Df** register has specified **top-of-page** output. When a display is output, it is also removed from the queue.

When the formatter reaches the end of a document (or the end of a section when section-page numbering is used), all displays still in the queue are automatically printed. This occurs before a **.SG**, **.TC** or **.CS** closing macro is processed.

The floating display registers and their settings and effects are as follows:

De

0 No special action occurs; this is the default value of **De**.

1 A page eject always follows the printing of each floating display. Only one floating display appears on a page, and no text follows it.

val For any other value entered, **mm** performs as though the value is 1.

.Df

0 Floating displays are not printed until the end of the document or at the end of the section if section-page numbering is used.

1 The new floating display is printed on the current page if there is room; otherwise, it is held until the end of the section or document.

2 Exactly one floating display is printed from the queue at the top of a new page (or at the top of a new column, when in two-column mode).

3 One floating display is printed at the top of the current page or column if there is room.

4 As many displays as will fit are printed starting at the top of a new page or column.

Note: If register **De** is set to **1**, each display is followed by a page eject, causing a new top of page for each display. This also applies to value **5**, which follows.

5 Print a new floating display on the current page if there is room. Print as many displays as will fit starting at the top of a new page or column. (See preceding note.)

val For any value greater than **5**, the action performed is the same as for the value **5**.

Note: The **.WC** (width-control) macro is also used to control handling of displays in double-column mode and can control the break in the text before floating displays (see "Two-Column Format" in topic 2.7.4.8.1).

2.7.4.11.3 Tables

The **.TS** (table start) macro and **.TE** (table end) macro let **mm** use the **tbl** preprocessor. They delimit the text to be examined by **tbl** by creating a ***table block*** and setting the proper spacing between it and the text which precedes or follows it.

The display functions and the **tbl** delimiting function are independent of one another. If you need to keep blocks together that contain a mixture of tables, equations, filled text, unfilled text, and caption lines, enclose the table block within a display. You can place table blocks within static displays or float them by placing them inside floating displays.

This is the sequence for the **.TS** and **.TE** macros:

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

The options, descriptors, title lines, and so forth, are detailed in Chapter 10, "Formatting Tables" in topic 2.10. The **.TS** and **.TE** macros also let you process tables that extend over several pages. The **H** parameter to the **.TS** macro causes a table heading to print for each page of the table.

Note: You must use the **.TH** macro when you use the **H** parameter.

The **.TH** macro specifies that a table header (title line) be printed on as many lines as required. (This macro is a memorandum macros feature, not a feature of **tbl** command.) You can specify the letter **N** as a parameter; this causes the table header to print only if it is the first table header on the page. This option is useful when you need to build long tables from smaller **.TS H/.TE** segments, for example:

```
.TS H
Title line
.TH
data
.TE
.TS H
Title line
.TH N
data
.TE
```

The table heading appears at the top of the first table segment, and no heading appears at the top of the second segment when both appear on the same page. However, the heading does appear at the top of each page that the table continues onto. This feature is useful when a single table must be broken into segments because of table complexity (for example, too many blocks of filled text). If each segment has its own **.TS H/.TH** sequence, each segment will have its own header. However, if each table segment after the first uses **.TS H** or **.TH N**, then the table header appears at the beginning of the table and at the top of each new page or column onto which the table continues.

Note: You can, for higher-resolution printing terminals, use the **-E** flag with the **mm** command (or the **-e** flag, of **nroff**, see 2.8.2.1). This may cause better alignment of features such as the lines forming the corner of a box. However, **-E** (**-e**) is not effective when **col** is used for postprocessing.

2.7.4.11.4 Equations

The equation preprocessor commands **neqn** and **eqn** (see "Command Lines" in topic 2.9.2) expect to use the **.EQ** (equation start) and **.EN** (equation end) macros as delimiters in the same way **tbl** uses **.TS** and **.TE**. Unlike a table block, however, equations within **.EQ** and **.EN** must occur within **.DS** and **.DE** (static display) delimiters:

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

Note: There is an exception to this rule: if you use **.EQ** and **.EN** only to specify the delimiters for an inline equation or to specify (**n)eqn define** statements (see Chapter 9, "Formatting Mathematics" in topic 2.9), do not use **.DS** and **.DE** display delimiters, or what was intended as inline equation output will be separated from preceding and following text by vertical space.

The **.EQ** macro takes a parameter that it uses as a label for the equation. By default, the label appears at the right margin next to the general equation, centered vertically. You can set the **Eq** number register to **1** to change the labeling to the left margin. (See also "Caption Macros" in topic 2.7.4.11.5.)

The equation is automatically centered for centered displays; otherwise, the equation is adjusted to match the margin opposite from the label.

2.7.4.11.5 Caption Macros

The **.FG** (figure title) macro, **.TB** (table title) macro, **.EC** (equation caption) macro, and **.EX** (exhibit caption) macro are normally used inside **.DS/.DE** pairs to automatically number and title figures, tables, and equations. They use the **Fg**, **Tb**, **Ec**, and **Ex** registers respectively to reset counters in sections (see "Setting Number Registers from the Command Line" in topic 2.7.2.4.5). You can change the format of the caption sequence numbers by using the **.af** formatter request (see topic 3.13.4).

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

Output 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. For example, the call:

```
.FG "This is an illustration."
```

yields:

Figure 1. This is an Illustration

Each macro provides an appropriate title preface. The prefatory caption macros are:

Macro:	**.FG**	**.TB**	**.EC**	**.EX**
Output:	**Figure.**	**TABLE.**	**Equation.**	**Exhibit.**

You can cause the label and sequence number to be separated by a dash instead of by a period by setting the **Of** (figure caption style) register from the default value of **0**, to **1**. For instance:

```
.nr Of 1
.EX "This is Exhibit A"
```

produces:

Exhibit 1 - This is Exhibit A

The **override** string is used to modify the normal numbering.

If **flag** is omitted or **0**, **override** is used as a prefix to the number.
If **flag** is **1**, **override** is used as a suffix.
If **flag** is **2**, **override** replaces the sequence number.

Note: When **section-figure numbering** is set by the **mm** command line flag **-rN5**, the **override** string is ignored.

As a matter of style, table headings are usually placed ahead of the text of the tables, while figure, equation, and exhibit captions usually occur after the corresponding figures, equations, and exhibits.

2.7.4.11.6 Lists of Figures, Tables, Exhibits, and Equations

You can have a list of figures, list of tables, list of exhibits, and list of equations printed. They are printed after the table of contents is printed if the **Lf**, **Lt**, **Lx**, and **Le** number registers (respectively) are set to **1**. **Le** is **0** by default, and the rest are set to **1** by default.

The titles of these lists are held in string registers that you can change by redefining the strings. The string registers are defined here with the strings they hold by default.

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

The printing of these lists are affected by the **Cp** (contents placement) register.

2.7.4.11.7 Constant Width

You can use the **.CW** and **.CN** macros with **cw** command to print in a constant width font. The **cw** command creates an output file containing **troff** requests to control text appearance by controlling local horizontal motion; the result when printed has the appearance of video display or line-printed output, with each character having the same (printed) character width. The constant width output may be in line with text in the prevailing font or imbedded as a display. A feature of **cw** can buffer constant width text so **troff** does not interpret requests and macros within a constant width display. This makes **cw** particularly useful for creating examples of text formatting.

You must use the **cw** preprocessing command to filter a text file that uses **.CW**/**.CN** pairs before processing the text through other preprocessors or the formatter. You can use **cw** before **tbl** or **eqn**, for example; if all three filters are used, you must use them in that order.

The memorandum macros and the viewgraph macros interpret the definitions of the five constant width macros in an appropriate context for the macro package. The five constant width macros are:

.CD [**parms**]
The **.CD** (change delimiters) macro changes delimiters and settings of **cw** command parameters within text instead of on the command line. It accepts any of the parameters of the **.cw** command.

.CN [**parms**]
The **.CN** (constant width end) macro marks the end of the text to be set in constant width font. It accepts the same parameters as the **cw** command.

.CP [**cw**] [**pfnt**] ...
The **.CP** (constant width parameters) macro concatenates the parameters that follow it as text; the odd-numbered parameters are set in constant width font, and the even-numbered parameters are in the prevailing font.

.CW [**parms**]
The **.CW** (constant width) macro marks start of text to print in constant width font. The parameters may be any parameters accepted by the **cw** command (see **cw**, in the *AIX Operating System Commands Reference*). **.CW** causes a break.

.PC pfnt [**cw**] ...
The **.PC** (reversed constant width parameters) macro works like **.CP**, except the even-numbered parameters are in constant width font and the odd-numbered parameters are in prevailing font.

Note: The examples in this manual are produced in a constant width font.

You can use the **checkcw** command to examine a file to assure that each **.CW** and **.CN** pair is matched. For further information about **cw** and **checkcw**, see the **cw** command in the *AIX Operating System Commands Reference*.

2.7.5 Typography

Type appearance features (typography) control the appearance of characters, fonts, and line spaces.

Subtopics
2.7.5.1 Character Accents
2.7.5.2 Fonts
2.7.5.3 Boxed Text
2.7.5.4 Vertical Spacing
2.7.5.5 troff Point Size and Vertical Spacing

2.7.5.1 Character Accents

You can use the following strings to produce accents for characters:

Accent	**Input**	**Output**
grave	**e***`	è
acute	e*'	é
tilde	n*(~)	n tilde
cedilla	c*,	ç
umlaut	u*:	ü
Umlaut	U*;	Ü
circumflex	o*(^)	ô

2.7.5.2 Fonts

If you are using **troff**, there are several fonts you can use to highlight your text. You have more limited choices using **nroff**. These macros control font selection:

```
.B [bld] [pfnt] ...
.I [itl] [pfnt] ...
.R
```

When called without parameters:

.B changes the font to bold (**nroff**: double-strike for many printers).

.I changes the font to italic (**nroff**: underline for many printers).

.R sets font to normal Roman font. The font stays as it is set until it is changed by a font change macro. For example:

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

This yields (using **troff**):

here is some text
here is some text

If you call **.B** or **.I** with one parameter, that parameter is printed in the corresponding selected font; then the previous font is restored. You can delimit the parameter in quotes to do a phrase, sentence, or paragraph with a single macro call:

```
This is
.B not
very difficult, but it
.I " could be " easier.
```

This results in (using **troff**):

This is **not** very difficult, but it *could be* easier.

If you give two or more parameters (to a maximum of **6**) to a **.B** or **.I** macro, the second parameter is then attached to the first with no intervening space (1/12 space, if the leading font is italic, in **troff**), but is printed in the previous font; and the remaining parameters are similarly alternated. For example:

```
.I italic " text " right -justified
```

This produces in **troff**:

italic text **right**-justified

These macros alternate with the prevailing font at the time they are called.

Only two fonts are held in memory. When one is switched out of the current font, it becomes the previous font, unless two fonts are selected.

When two fonts are selected, any previous font is forgotten. To alternate specific pairs of fonts, the following macros are available:

.BI Sets bold to current font and italic to previous font.
.BR Sets bold to current font and Roman to previous font.
.IB Sets italic to current font and bold to previous font.
.IR Sets italic to current font and Roman to previous font.
.RB Sets Roman to current font and bold to previous font.
.RI Sets Roman to current font and italic to previous font.

Note: Font changes in headings are handled separately (see "Altering Appearance of Headings" in topic 2.7.6.2).

If you are using a printing device that cannot underline, for instance, you may wish to undefine (cause the formatter to remove) the request at the beginning of the document to eliminate all underlining.

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

If you are using a printer with selectable fonts and wish to work around font limitations, you can use the **.RD** macro to cause the printer to pause for keyboard input each place you need to change fonts. Do not attempt to change the pitch this way, because the right margin and perhaps vertical spacing are affected. (The results of this technique are device-dependent.)

2.7.5.3 Boxed Text

You can create a special text appearance using the **.BX** (boxed text) macro:

```
.BX parm1 [parm2]
```

.BX encloses **parm1** (which is set in bold type) in a box, appending **parm2**, if any, to the box without separating spaces. For example:

```
.BX EXIT
```

produces:

```
+----------+
¦ EXIT     ¦
+----------+
```

2.7.5.4 Vertical Spacing

You can use the **.SP** (vertical spacing) macro to change vertical spacing in **troff**. **nroff** ignores this macro, deriving vertical spacing from the terminal type definition.

```
.SP [num]
```

The **num** parameter is an unscaled positive number that selects the number of lines to space down. If **num** is not specified, the default is one line. You can specify a decimal fraction for the **num** parameter.

The **.SP** macro does not allow the accumulation of vertical space by successive macro calls. Successive **.SP** calls produce the result of the largest single **.SP** parameter, not the sum of their parameters. The following sequence only produces three blank lines:

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

The **mm** command has many macros that use **.SP** for setting vertical spacing, so ignoring accumulated spacing is useful. For example, **.LE 1** immediately followed by **.P** produces only a half vertical space between the end of the list and the following paragraph (one line, in **nroff**).

The **.sp** request spaces the number of lines specified, unless **.ns** (no space) mode is set, in which case the request is ignored. (This mode is set at the end of a page header to eliminate spacing being caused by a **.sp** or **.bp** request occurring at the top of a page.) Both **.sp** and **.SP** are inhibited by the **.ns** request. You can turn this mode off using the **.rs** (restore spacing) request.

2.7.5.5 troff Point Size and Vertical Spacing

In **troff**, the default point size, obtained from the **mm** command register **S** (see "Setting Number Registers from the Command Line" in topic 2.7.2.4.5), is 10 points and the vertical spacing is 12 points. This results in output of 6 lines per inch. You can change the point size and vertical spacing by invoking the **.S** (size) macro:

.S [[±]**psize**] [[±]**vspacing**]

where point size and vertical spacing are expressed as numeric values.

If a parameter is negative, the current value is decreased by th number of points.
If a parameter is positive (signed), the current value is increased b the specified number of points.
If a parameter is unsigned, it is absolute and is used as the ne value.
.S without parameters defaults to previous value.
If the first parameter is specified but the second is not, then th default value is used for the vertical spacing, which is always two points greater than the current point size. (Footnotes are two points smaller than the body, with an additional three point space between footnotes.) A null ("") value for either parameter keeps the current value. (In the following table, **num** is an unsigned numeric value.)
If you give a **num** value for type size that is not supported by **troff**, it uses the next smaller size of type available.
If you give a **psize** value greater than 99, the default point size (10 points) is restored.
If you give a **vspacing** value greater than 99, the default vertical spacing (current point size plus 2 points) is used. For example, **.S 100** results in a type size of 10 points and a vertical spacing of 12 points. The **.S 14 117** macro call results in a type size of 14 points, and a vertical spacing of 16 points.

Table 7-3. .S Macro and Parameters

Call Configuration	**Point Size**	**Vertical Spacing**
.S	Previous value	Previous value
.S **"" num**	Current value	**num**
.S **num ""**	**num**	Current value
.S **n**	**num**	**num +2**
.S ""	Current value	Current value **+2**
.S "" ""	Current value	Current value
.S **num num**	**num**	**num**

The **.SM** (smaller string) macro lets you reduce the type size of a string

by one point:

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

If the third parameter is omitted, the **first** parameter is made smaller and is linked (without separating spaces) with the second parameter, if the latter is specified.

If all three parameters are present (even if any are null), the secon parameter is made smaller, and all three parameters are linked together, for example:

Input	**Output**
.SM R E D	RE D
.SM SOCKEYE	SOCKEYE
.SM PIN K	PINK
.SM (KING)	(KING)
.SM C OHO ""	COHO

2.7.6 Headings

You can use headings to provide useful organization to text. The memorandum macros let you control heading appearance, including optional heading numbering style, font control, and page break control. You may have up to seven levels of headings; level 1 is the highest level, and level 7 is the lowest.

The **.H** macro provides numbered headings. This is how the macro is used:

```
.H level [heading] [subheading]
.P
zero or more lines of text
```

The **subheading** is appended to the **heading**. It appears on the heading line but the formatter ignores it when making a Table of Contents.

The **.HU** (unnumbered heading) macro is a special case of the **.H** (heading) macro. 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 be at the hierarchy level given by register **Hu**, with an initial value of **2**. Thus, in the normal case, the only difference between:

```
.HU "headingtext"
.P
zero or more lines of text
```

and:

```
.H 2 "headingtext"
.P
zero or more lines of text
```

is the printing of the heading mark for the latter. Both have the effect of incrementing the numbering counter from level 2, and resetting to zero the counters for levels 3 through 7. **.HU** can be especially helpful for setting up special sections that do fit well into the numbering scheme of the main body of your document.

Note: Strictly speaking, you do not need a **.P** paragraph macro immediately after a **.H** or **.HU** heading for an unnumbered heading. These macros also perform the function of the **.P** macro, and an immediately following **.P** is ignored (see "Vertical Spacing" in topic 2.7.5.4). It is consistent and common practice to lead each paragraph with a **.P** macro, however, to assure that all paragraphs throughout a document have a uniform appearance.

Subtopics

2.7.6.1 Heading Appearance

The normal appearance of headings set with the **.H** macro varies with the heading level. (See "Numbered Paragraphs" in topic 2.7.7.2.) First-level headings are preceded by one vertical space; all others are preceded by a half vertical space (one line in **nroff**).

.H 1 **"heading"**
: Gives a bold heading followed by a half vertical space (one line, in **nroff**). Following text begins on a new line and is indented according to the current paragraph type.

.H 2 **"heading"**
: Yields a bold heading followed by a half vertical space (one line, in **nroff**). The following text begins on a new line and is indented according to the current paragraph type.

.H **num "heading"**
: Produces an underlined (in **nroff**, but italic in **troff**) heading followed by two blank spaces for levels **3** through **7**. The following text appears on the same line with the heading.

Appropriate numbering and spacing (horizontal and vertical) occur even if the heading text is omitted from an **.H** or **HU** macro call.

2.7.6.2 Altering Appearance of Headings

You can modify the appearance of headings quite easily by setting registers and strings at the beginning of the document. This permits quick alteration of the document style, because style-control information is concentrated in a few lines rather than being distributed throughout the document. You can control whether headings are preceded by a page break, spacing before and after headings, centering, typeface, type size, and numbering and marking styles. 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 following number registers offer this control:

Ej The **Ej** (eject) register holds a value that indicates the level of heading that causes a new page. Any heading level equal to or lower than the value in **Ej** causes a new page. A request of **.nr Ej 2**, for example, causes level 1 or 2 headings to start on a new page.

Hb The **Hb** (heading break) register holds a value that determines the heading level below which a break occurs after the heading text. (A heading that is not followed by text on the line is a ***standalone heading***.) For example, the request **.nr Hb 3** causes heading levels 4 through 7 to have text follow the heading on the same line. The default value of **Hb** is 2.

Hc The **Hc** (heading centering) register holds a value that is used to determine whether to center headings. If a heading level is less than or equal to the value in **Hc**, and if it is also a standalone heading, it is centered. The value of **Hc** is initially **0**, resulting in no centered headings.

HF The **HF** (heading font) string register contains a string of up to seven digits, separated by spaces, that consecutively specify the font for each heading level, one through seven. The default value stored in the **HF** string register is the same for both **nroff** and **troff**, though what **nroff** interprets as an **underline** specification, **troff** treats as an **italic** specification. The default value of **HF** is **3 3 2 2 2 2 2**. The numeric codes you can use are:

1 specifies Roman font. (For **nroff**, this is whatever basic font the printer uses.)
2 specifies italic (**nroff** underline) font.
3 specifies bold font.

Thus, the default font for levels 1 and 2 are bold, and levels 3 through 7 are italic in **troff** and underlined in **nroff**.

You can easily define a new string in the **HF** string register using the **.ds** formatter request. You can use the numerals 1, 2, and 3; if you do not provide seven digits, **mm** assumes a value of 1 for the ungiven numerals. For example, you might enter:

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

The formatter will prepare headings as though the string value is **3 3 3 3 3 1 1**, and print heading levels 1 through 5 in boldface and levels 6 and 7 in Roman font.

.cu The **.cu** (continuous underline) request underlines all characters, including spaces and tabbed space. By

default, the **mm** command attempts to use the continuous underline 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 too long for a single line, **mm** uses the **.ul** request instead of the **.cu** request; only letters and digits are underlined.

.ul You can use the formatter **.ul** (underline) request to underline visible heading characters. See also the **.cu** request, and the **nroff -rU** flag, which lets you set heading underlining from the command line (see topic 2.7.2.4.5). You can also set **nroff** heading underlining by changing the value stored in the **U** number register from **0** to 1.

Hi The **Hi** (post-heading indent) register holds a value that determines the indentation of text following a standalone heading. For any standalone heading (see **Hb**, preceding), the alignment of the next line of output is controlled by the **Hi** register value:

If **Hi** is **0**, text is left-justified.
If **Hi** is 1 (the default value), the text is indented according to the paragraph type as specified by the registers **Pt**, **Pi** and **Ps**.
If **Hi** is 2, text is indented to line up with the first word of the heading itself, causing the heading number to stand out more clearly.

The **Hs**, **Hi**, and **Hb** registers work together to let you tailor your headings. For example, to cause:

A half vertical space to appear after the first three heading levels (**troff**)
No run-in headings
Text following all headings to be left-justified (regardless of the value of **Pt**)

place the following at the top of the document:

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

.HM The **.HM** (heading mark) macro accepts up to seven parameters that set the type of numbering or marking for each of the corresponding seven heading levels. The type of marking for each level is stored as a value to a number register. The number registers are named **H1** through **H7** with **H1** representing the level 1 heading, **H2** the level 2 heading, and so on.

The headings for each level are normally marked with Arabic numerals, but **.HM** lets you change the values stored in the various registers all at the same time. You could use the **.nr** request to change the value in a register, but the values of each of these registers is affected by the value stored in its lower-numbered counterparts. **.HM** lets you tailor an outline or document style for your needs.

Value Interpretation

1	Arabic (default for all levels).
0001	Arabic with a fixed number of digits. Numbers are padded as needed with leading zeros.
A	Uppercase alphabetic.
a	Lowercase alphabetic.
I	Uppercase Roman.
i	Lowercase Roman.

Legal values for registers **H1** through **H7** are: Omitted values are interpreted as 1, but illegal values will not change a register.

By default, **mm** gives a complete heading mark for a given level, marking the levels by showing each level separated by periods (or decimal points). For example, a level 4 heading could be marked as:

2.4.2.2 Level Four Heading.

You can cause all the previous level numbers to be omitted by changing the value in the **.Ht** (heading-mark type) register.

HP The **HP** (heading point) string register holds a value that lets you specify the point size of the heading text by giving a string of numeric point size indicators (that can be relative or absolute) separated by spaces. For example:

```
.ds HP 12 11 10 10 10 10 10
```

sets the level 1 heading at 12, the level 2 heading at 11, and subsequent levels at 10.

Notes:

1. You must specify the relative (**troff**) point sizes as relative to body text point size. If you use a default value heading point size, then the default heading type size is used. The default heading type size and body type size are not necessarily the same. Standalone headings are reduced one point from the text size while run-in headings are not.

2. The **HP** string register only affects the point size of the headings; no adjustment is made for vertical spacing (see "User-Defined Headings" in topic 2.7.6.5).

Hs The **Hs** (heading space) register establishes the vertical space following the heading. These spaces serve to separate the heading from the text on those heading levels that cause a break. The default value is **2**, which sets spacing to one full vertical space.

Ht The **Ht** (heading mark type) register holds a value that indicates whether the heading mark is indicated in full or truncated to the last level. The default value is **0**, which results in full marking. Changing this value to 1 causes only the last level to be output, followed by a period, space(s) and the heading text.

When truncation is used, the heading is not indented by the length of the truncated heading mark. Therefore, decimal alignment is not kept if truncation is used for Arabic-numbered headings.

2.7.6.3 Table of Contents

You can cause the text of headings and their corresponding page numbers to be automatically collected for a table of contents using the **.TC** (Table of Contents) macro. This is accomplished by:

1. Specifying in the **Cl** (contents-level) register what level headings are saved.
2. Calling the **.TC** macro at the end of the document.

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

Note: Due to the way the headings are saved, it is possible to exceed the formatter storage capacity, especially when saving many levels of many headings while also processing displays and footnotes (see "Displays" in topic 2.7.4.11 and "Footnotes" in topic 2.7.9). If this happens, the **Out of temp file space** diagnostic message is issued. Your only remedies are to use fewer levels for the Table of Contents, reduce or eliminate displays, or to have fewer words in the heading text.

You can use this macro only once per document. There is no table of contents for a letter or general style document.

Subtopics
2.7.6.3.1 Table of Contents Macro
2.7.6.3.2 Contents Registers
2.7.6.3.3 Contents Indent Register

2.7.6.3.1 Table of Contents Macro

The parameters to the **.TC** macro control the spacing before each entry, the placement of the associated page number, and additional text on the first page of the table of contents before the word **CONTENTS**.

.TC [**slevel**] [**spacing**][**tlevel**] [**tab**] [**hdg1** ... **hdg5**]

Spacing before each entry is controlled by the first two parameters headings that have a level less than or equal to **slevel** will have **spacing** blank lines (in **troff**, halves of vertical spaces) before them. Both **slevel** and **spacing** default to 1. This means that first-level headings are preceded by one blank line (a half vertical space, in **troff**).

The third and fourth parameters control the placement of the pag number for each heading. You may justify the page numbers at the right margin with either blanks or dots (leaders) separating the heading text from the page number, or the page numbers can follow the heading text.

- For headings that have a level less than or equal to **tlevel** (the default is **2**), the page numbers are justified at the right margin. In this case, the value of **tab** determines the character used to separate the heading text from the page number.
  -- If **tab** is **0** (the default value), a series of dots (periods) are used. (This sequence of dots is called a ***leader***.)
  -- If **tab** is greater than **0**, spaces are used.
- For headings that have a level greater than **tlevel**, the page numbers are separated from the heading text by two spaces. This produces a ***ragged right*** margin, since the page numbers do not align along the right margin.

All additional parameters (text strings for lines of a table o contents page header), if any, are horizontally centered on the page and precede the actual table of contents. You may have up to five lines of header text, specified by the macro parameters or by one of two user-exit macros.

- When you do not give more than four parameters to **.TC**, you may define either the **.TX** or **.TY** (table of contents user-exit) macros.
  -- Defining the **.TX** macro gives you some defined header text treatments, followed by the heading **CONTENTS** and then the Table of Contents. For example, the following input:

```
.de TX
.ce 2
Special Application\*(EMMessage Transmission
.sp 2
.in +10n
Approved: \1'3i'
.in
.sp
..
.TC
```

yields:

Special Application-Message Transmission

Approved: ----------

CONTENTS
.
.
.

-- Defining the **.TY** macro gives you similar header treatments but suppresses the word **CONTENTS**. If you define the **.TY** macro as an empty macro, the line **CONTENTS** is suppressed, with no text given:

```
.de TY
..
```

- If you give five or more parameters to **.TC**, the fifth through ninth parameters are taken as lines of header text. The user-exit macro **.TX** is called without a parameter following the last parameter-defined header line, resulting in given header lines followed by a line with the centered word **CONTENTS**, then by the table of contents.

2.7.6.3.2 Contents Registers

The default value of the **Cl** register is **2**, indicating that the first two levels of headings are saved.

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

The **Oc** (contents organization) register controls the numbering method of the table of contents pages. (It is your responsibility to give an appropriate page footer setting to place the page number. Usually the same **.PF** used in the body of the document is used unchanged. See "Page Footer" in topic 2.7.4.3.2.)

- By default (value **0**), the table of contents pages have lowercase Roman numeral page numbering.
- If you set the **Oc** register to **1**, the **.TC** macro does not print any page number but instead resets the **P** (page number) register (see "Setting Number Registers from the Command Line" in topic 2.7.2.4.5) to **1**.

The **Cp** (contents placement) register determines whether lists of figures, tables, exhibits, and equations (see "Lists of Figures, Tables, Exhibits, and Equations" in topic 2.7.4.11.6) are given on the same page with the table of contents, or each list is placed on a separate page. An empty list will not appear on output.

- The default value is **0**, which results in each list receiving a separate page.
- You may set **Cp** to one, causing these lists to appear on the same page as the table of contents.

2.7.6.3.3 Contents Indent Register

By default, the first level headings appear in the table of contents at the left margin. Subsequent levels are aligned with the text of headings at the preceding level. These indentations are changed by defining contents of the **Ci** (contents indent) string register, which takes up to seven scaled indentation parameters corresponding to the heading levels.

Note: You must give at least as many **.Ci** parameters as there are heading levels represented in the **Cl** register, and the parameters must be scaled.

2.7.6.4 Level 1 Headings and Page Numbering Style

By default, pages are numbered sequentially at the top of the page. For large documents, you may wish to use ***section-page numbering***. The ***section*** is the number of the current level 1 heading, and the section pages are numbered sequentially within each section beginning at page 1; page numbering occurs at the bottom of the page rather than the top.

This page numbering style is achieved by specifying the **-rN3** or **-rN5** flag on the command line (see topic 2.7.2.4.5). As a side effect, this also has the effect of setting the value of the **Ej** number register to **1** so each section begins on a new page. In this style, the page number is printed at the bottom of the page.

Though page numbering is reset for each section under this style, footnote numbering is not. It remains cumulative for the entire document.

2.7.6.5 User-Defined Headings

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

You can use the **.HX**, **.HY**, and **.HZ** (user-defined heading) macros to obtain a final degree of control over the **mm** heading mechanisms. These macros are left undefined by memorandum macros; they are intended for you to define. If you do not define these macros, **mm** does not use them and all the normal heading control activities occur. If the **.HX**, **.HY** or **.HZ** macros are defined, the definitions are applied at appropriate places in the processing by the **.H** macros. These user-defined macros can apply to all headings.

You define these macros with these parameters:

.H**X derived-level real-level**

This is the general sequence of how the user-defined heading macros work when they are defined:

1. The **.H** macro (or its special case **.HU**) is called. **.H** then:

   Increments the heading counter of the specified level (see "Altering Appearance of Headings" in topic 2.7.6.2).
   Produces vertical space to precede the heading (see topic 2.7.6.2).
   Collects the heading mark, the string of digits, letters, and periods needed for a numbered heading (see "Heading Appearance" in topic 2.7.6.1).

2. **.H** (or **.HU**) calls **.HX**, and then:

   After **.HX** is called, the size of the heading is calculated. This other processing causes the loss of certain features that may have been included in **.HX**; such as, **.ti** for temporary indent.

   .HY is called so you can respecify these features if you need them.

   Other heading processing is performed; the heading is printed.

   .H (**.HU**) calls **.HZ** as its last action, after the heading is printed.

3. When **.HX** is called, all normally accessible number registers and string registers can be referenced, as well as the following reserved number registers:

   }0 If **real-level** is:

   Nonzero, this string register contains the heading mark string. Two constant spaces (to separate the mark from the heading) are appended to this string.
   0, this string is **null**.

   ;0 This register sets the vertical spacing to follow the heading.

   A value of **0** means the heading is run-in.
   A value of 1 means a break (but no blank line) is to follow

the heading (register **Hb** is in effect).
A value of 2 means that a half vertical space is to follow the heading (one line, in **nroff**). (Register **Hs** is in effect).

}2 If register **;0** value is **0**, this string register contains a string of two constant spaces that are used to separate a run-in heading from the following text. If register **;0** is nonzero, this string register holds a null string.

;3 This register holds an adjustment factor for a formatter need request **.ne** issued before the heading is actually printed. On entry to **.HX**, it has the value of 3 if **derived-level** equals 1 and has a value of 1 otherwise. The **.ne** request is for the sum of the following lines:

The value in the register **;0** taken as blank lines (halves of vertical spaces, in **troff**).
The value in the register **;3** taken as blank lines (halves of vertical spaces, in **troff**).
The number of lines for the heading.

You can alter the values of **}0**, **}2**, and **;3** within **.HX** as you wish. The following are examples of actions you might perform by defining **.HX** to include lines (for clarity, the &ballot. represents a space character):

Call **Result**

.if \\$1=1 .ds }0 \\n(H1.0\&ballot.\&ballot.
Change level 1 heading mark from format **num.** to **num.0**.

.if \\n(;0=0 .ds }2 .\&ballot.\&ballot.
Separate a run-in heading from the text with a period and two constant spaces.

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

.if \\$1=1 .sp 3
Add three additional blank lines (halves of a vertical space, in **troff**) before each level 1 heading.

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

If temporary string registers or macros are used within **.HX**, choose their names with care.

.HY is called after the **.ne** is issued. You must repeat certain features requested in **.HX**. For example:

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

.HZ is called at the end of **.H** to allow user-controlled actions after the heading is produced. In a large document, for example, sections may correspond to chapters of a book, and you may want to change a page header or footer:

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

2.7.7 Paragraphs

You can use paragraph macros **.P** and **.nP** to begin a paragraph and control paragraph style. A paragraph macro immediately precedes the text of a paragraph.

Subtopics
2.7.7.1 Block and Indented Paragraphs
2.7.7.2 Numbered Paragraphs
2.7.7.3 Paragraph Separation

2.7.7.1 Block and Indented Paragraphs

There are two forms of unnumbered paragraphs you can select using the **.P** macro. The request form is:

```
.P type
one or more lines of text.
```

If you do not give **type**, the default paragraph style is used, in which **type** is **0**. This gives a simple left-justified paragraph with no indentation of the first line (beyond the established page offset).

If **type** is **1**, the style is set for an indented paragraph. The first line of the paragraph is indented five spaces, while the rest of the lines are left-justified at the page offset.

The default style is controlled by the register **Pt**. The initial value of **Pt** is **0**, which always provides left-justified paragraphs; to achieve indentation, you insert **.P** before each paragraph that does not contain a heading. You can force all paragraphs to be indented by setting the value of the number register **Pt** as 1 at the beginning of the document:

```
.nr Pt 1
```

If you set the value of **Pt** to 2, paragraphs are indented except after headings, lists, and displays.

The amount a paragraph is indented is contained in the register **Pi** with a default value of 5. Of course, both **Pi** and **Pt** register values must be greater than zero for any paragraphs to be indented. Make sure you set registers that control indentation at the beginning of the document.

Note: Register values that specify indentation must be unscaled and are treated as character positions, a number of ***ens***. In **nroff**, an en is the width of a normal character, typically 2/3 of point size. In **troff**, an en is a variable scaled value.

You can force a left-justified or indented paragraph regardless of the value of **Pt**. The request **.P** always forces left justification; the request **.P 1** always causes indentation by the amount specified by the register **Pi**.

If **.P** occurs inside a list, the indent (if any) of the paragraph is added to the current list indent (see "Lists" in topic 2.7.8).

2.7.7.2 Numbered Paragraphs

You can cause numbered paragraphs by setting the register **Np** to 1 to produce paragraphs numbered within first level headings (see "Heading Appearance" in topic 2.7.6.1). The head numbers will look like:

```
1.01, 1.02, 1.03, 2.01 ...
```

You can obtain a different style of numbered paragraphs by using the **.nP** macro rather than the **.P** macro for paragraphs. This produces paragraphs that are numbered within second level headings and contain a ***double-line indent*** in which the text of the second line is indented and aligned with the text of the first line so that the number stands out:

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

2.7.7.3 Paragraph Separation

The amount of vertical spacing used between paragraphs is stored in number register **Ps**. By default, **Ps** is set to **1**, which in **troff** is a half vertical space, and in **nroff** is one line.

Note: In paragraph separation, lists, displays, and other places where **troff** interprets a separation as a half vertical space, **nroff** interprets the separation as one line.

2.7.8 Lists

This section describes many different kinds of lists: automatically numbered and alphabetized lists, bulleted lists, dashed lists, lists with arbitrary marks, and lists starting with arbitrary strings, such as terms or phrases to be defined.

Subtopics

2.7.8.1 Basic Lists

To avoid repetitive typing of parameters to describe the appearance of items in a list, the **mm** command provides you a convenient way to specify lists. All lists have the following parts:

A ***List start*** macro that controls these features of the list:

- Line spacing
- Indentation
- Marking with special symbols
- Numbering or alphabetical labeling.

These are the list start macros:

.AL (***alphabetical list***) macro.
.BL (***bullet list***) macro.
.DL (***dash list***) macro.
.ML (***marked list***) macro.
.RL (***reference list***) macro.
.VL (***variable-item list***) macro.
.LB (***list-begin***) macro that all other list start macros use and that you can use to create a custom list layout.

One or more occurrence of the **.LI** (list item) macro, each use followed by the actual text of the corresponding list item. The characteristics of list item macros are the same in all types of lists (see "List Items" in topic 2.7.8.3).

The **.LE** (list end) macro that ends the list and restores the previous list status information (see "List End" in topic 2.7.8.4).

You can nest lists up to six levels. The **.LB** macro saves previous list status information (indentation, marking style, and so on); the **.LE** macro restores it. You specify the format of a list only once by the list-start macro that you use. By building on the existing structure, you can create your own customized sets of list macros (see "List Begin and Customized Lists" in topic 2.7.8.6).

2.7.8.2 List Start

The following are the various list-start macros. Each list start macro is actually implemented as a call to the basic **.LB** macro (see "List Begin and Customized Lists" in topic 2.7.8.6).

Subtopics
2.7.8.2.1 Automatically Numbered or Alphabetized Lists
2.7.8.2.2 Bullet List and Dash List
2.7.8.2.3 Marked List
2.7.8.2.4 Reference List
2.7.8.2.5 Variable-Item List

2.7.8.2.1 Automatically Numbered or Alphabetized Lists

You can use the **.AL** (alphabetical list) macro to begin sequentially numbered or alphabetized lists. If you do not give parameters, the list is numbered and the text is indented by the value stored in the **Li** register, initially 6, from the indent in force when the **.AL** is called. The value is an unscaled value representing the number of ens to indent. The value of 6 leaves enough room for a space, two digits, a period, and two spaces preceding the text.

You call the macro in this form:

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

You can give the **type** parameter to set the type of sequence marking. Its value must be 1, A, a, I, or i.

Note: The **0001** format is not permitted, so you may not select the padded Arabic number marking style using this list start macro.

If **type** is omitted or null, the value **I** is the default, giving uppercase Roman marking. If **text-indent** is nonnull, it is used as the number of spaces from the current indent to the text, instead of using the value of **Li** for the list only. If **text-indent** is null, then the value of **Li** is used.

The value of **text-indent** as a second parameter is the number of ens that following text lines are indented from the marker. It can be a signed or unsigned number, and the result is an ***overhang*** (the following line begins to the left of the marker) if a negative number is given.

If you give a numeral **1** as a third parameter, the list is output as a ***compact list***; a half vertical space (one line, in **nroff**) does not separate the items in the list. A blank line still occurs before the first item, however.

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

```
.nr Ls 0
```

specifies that spacing will not occur around any list items. The default value for **Ls** is **6**, which is the maximum list nesting level.

2.7.8.2.2 Bullet List and Dash List

.BL begins a ***bullet list***, in which each item is marked by a bullet ( ) followed by one space.

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

If **text-indent** is nonnull, it overrides the default indentation, which is the paragraph indentation value given in the register **Pi** (see "Block and Indented Paragraphs" in topic 2.7.7.1). In the default indentation, the text of bullet (and dash) lists line up with the first line of indented paragraphs, leaving the marker exposed.

If you give the numeral **1** as the second parameter, the list is a compact list.

The **.DL** (dash list) macro is identical in use to **.BL** macro, except that a dash is used instead of a bullet.

2.7.8.2.3 Marked List

The **.ML** is much like **.BL** and **.DL** macros, but you can specify an arbitrary **mark**, which may consist of more than a single character. You call the marked list macro like this:

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

The **text-indent** parameter sets the number of ens to indent. If you do not specify the indent value, the text is automatically indented one more space than the width of **mark**. You can give a null value for **text-indent**. If you specify the numeral **1** as a third parameter, The result is a compact list; otherwise, the usual half vertical space (one line, in **nroff**) separates each item of the list.

Note: The **mark** must not contain ordinary (expandable) spaces, because item alignment is lost if the right margin is justified (see "Constant Spaces and Character Translation" in topic 2.7.2.5.3).

2.7.8.2.4 Reference List

The **.RL** (reference list) macro begins an automatically numbered list in which the list item numbers are enclosed by **[ ]** square brackets. The form of the macro call is:

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

You can supply a **text-indent** parameter, as for **.AL**. If the text indent is omitted or null, its default value is **6**, which is convenient for two-digit lists. If you supply a numeral **1** as a second parameter, the result is a compact list.

2.7.8.2.5 Variable-Item List

When a list begins with a **.VL**, there is effectively no current mark; it is expected that each list item (**.LI**) will provide its own mark. The (variable-item list) macro is called in this form:

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

The variable-item list is typically used to display definitions of terms or phrases.

The **mark-indent** parameter gives the number of en spaces from the current indent to the beginning of the **mark**, and it defaults to **0** if omitted or null.

The **text-indent** parameter gives the distance from the current indent to the beginning of the text. If the numeral 1 third parameter is specified, the result is a compact list.

The following is an example of **.VL** usage:

```
.tr ~
.VL 15 2
.LI First~Mark
Here is a description of the first mark;
``First~Mark:'' of the .LI line
contains a tilde translated to a
constant 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 a constant
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
```

the output of which is:

First Mark — Here is a description of the first mark; "First Mark" of the **.LI** line contains a tilde translated to a constant space in order to avoid extra spaces between "First" and "Mark".

Second Mark — This is the second mark, also using a tilde translated to a constant 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 parameter on the last **.LI** above is required; otherwise a hanging indent would have been produced. You produce a hanging indent by using **.VL** and calling **.LI** with no parameters or with a null first parameter. 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 ens.
.LE
```

the output of which is:

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

Note: Do not put ordinary (expandable) spaces in **mark**, because item alignment is lost if the right margin is justified (see "Constant Spaces and Character Translation" in topic 2.7.2.5.3).

2.7.8.3 List Items

The **.LI** (list item) macro is used with all lists. It normally causes the output of a single blank line (a half vertical space, in **troff**) before the item, although you can suppress this. The form of a list item macro is:

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

If you do not give parameters, **mm** labels the item with the current ***mark***, which is a mark specified by the most recent list start macro.

If you give a single parameter to **.LI**, that parameter is output instead of the current mark.

If you give **1** for a second parameter, the first parameter becomes a prefix to the current mark, thus letting you additionally emphasize one or more items in a list. (The second parameter can be **0** or **1**, with **0** being the default.) One constant space is inserted between the prefix and the mark. An example using the bullet list macro to make a ***bullet*** the current mark:

```
.BL 6
.LI
This is a simple bullet item.
.LI +
This replaces the bullet with a "plus."
.LI + 1
But this uses "plus" as a prefix to the bullet.
.LE
```

yields:

```
            This is a simple bullet item.

        +   This replaces the bullet with a "plus."

      +     But this uses "plus" as a prefix to the bullet.
```

Note: The mark must not contain ordinary (expandable) spaces, because alignment of items is lost if the right margin is justified (see "Constant Spaces and Character Translation" in topic 2.7.2.5.3).

2.7.8.4 List End

The **.LE** (list end) macro restores the state of the list back to the conditions that existed just before the most recent list-start macro call. This feature allows you to nest lists within lists (to six levels) in an orderly fashion. The form of the macro is:

```
.LE [1]
```

If the **1** flag is given, the **.LE** outputs a half vertical space (one line, in **nroff**). You generally use the flag only when the **.LE** is followed by running text, but not when followed by a macro that produces vertical space of its own; such as, **.P**, **.H**, or **.LI**.

2.7.8.5 Nested Lists

The input and corresponding output for several lists follow. The **.AL** macro call which gives alphabetized or sequentially numbered lists and the **.DL** dash list macro call are examples of list start macros.

Input:

```
.P
John, you need to get the equipment ready for the expedition.
.AL A
.LI
The truck needs some work.  It is overdue for general maintenance,
according to the service schedule.
There are some extra items to fix.
.AL
.LI
The exhaust system was damaged when you slid off the road
during that storm and got it hung up on rocks.
.DL
.LI
The muffler is mangled. Get a new one, but not one of those cheap ones.
I want the truck to be quiet, but too much exhaust restriction will
cause loss of power.
.LI
Make sure the mechanic checks for exhaust leaks after the muffler
is installed. I think the seal between the exhaust manifold and
downpipe may be broken.
.LE
.LI
Make sure the air conditioner is serviced.
.LE
.LI
The generator motor sounded pretty rough
by the end of our last trip. Get the maintenance
shop to do a tune-up on it. Get a couple of spare air filters
this time, too.
.LE
.P
Some of the students in my Archeology 301 class
have promised to come out and assist during spring break.
We need more shovels and sieves in case they show up.
```

Output:

John, you need to get the equipment ready for the expedition.

A. The truck needs some work. It is overdue for general maintenance, according to the service schedule. There are some extra items to fix.

    1. The exhaust system was damaged when you slid off the road during that storm and got it hung up on rocks.

        -- The muffler is mangled. Get a new one, but not one of those cheap ones. I want the truck to be quiet, but too much exhaust restriction will cause loss of power.

        -- Make sure the mechanic checks for exhaust leaks after the muffler is installed. I think the seal between the exhaust manifold and downpipe may be broken.

    2. Make sure the air conditioner is serviced.

B. The generator motor sounded pretty rough by the end of our last trip. Get the maintenance shop to do a tune-up on it. Get a couple of spare air filters this time, too.

Some of the students in my Archeology 301 class have promised to come out and assist during spring break. We need more shovels and sieves in case they show up.

2.7.8.6 List Begin and Customized Lists

The list-start macros suffice for almost all cases (see "List Start" in topic 2.7.8.2). However, if necessary, you can obtain more control over the layout of lists by using the basic **.LB** (list begin) macro, which is used by each of the list start macros. You call the macro this way:

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

Its parameters are as follows:

The **text-indent** parameter is a numeric value that gives the number of spaces for indenting text from the current offset and indent. Normally, this value is supplied by the register **Li** for automatic lists and by the register **Pi** for bullet and dash lists.

The combination of the values of **mark-indent** and **pad** determines the placement of the mark. The **mark-indent** parameter is typically set to **0**. The mark is placed within an area (called the ***mark area***) that starts **mark-indent** number of en spaces to the right of the current indent and ends where the text begins (that is, it ends **text-indent** spaces to the right of the current indent). Within the mark area, the mark is left-justified if you set **pad** to **0**.

If **pad** is some value greater than **0**, the number of en blanks that equal the value are appended to the mark, and the **mark-indent** value is ignored. The resulting string immediately precedes text on the same line. That is, the mark is effectively right-justified by **pad** spaces immediately to the left of the text.

The value of **type** and the character(s) held in **mark** interact to control the type of marking used. If you give a value of **0** for **type**, simple marking is performed using the marker character(s) found in the optional **mark** parameter.

If **type** is greater than **0**, automatic numbering or alphabetizing is performed, and **mark** is then interpreted as the first item in the sequence to be used for numbering or alphabetizing; the sequence is set as one of: **1**, **A**, **a**, **I**, or **i**.

- If **type** is **0** and **mark** is:

  -- Omitted, the result is a hanging indent.
  -- **string**, then **string** is the mark.

- If **type** is greater than **0**, and **mark** is:

  -- Omitted, the result is Arabic numbering.
  -- **1**, **A**, **a**, **I**, or **i**, the result is appropriate automatic numbering or alphabetic sequencing.

Each non-zero value of **type** from **1** to **6** selects a different way of displaying the marks. The following table shows the output appearance for each value of **type**, where **mark** is the generated number or letter.

Table 7-4. Displayed Style for List Mark

Type:	1	2	3	4	5	6

Appearance:	**mark**.	**mark**)	(**mark**)	[**mark**]	<**mark**>	[**mark**]

Note: The **mark** must not contain ordinary (expandable) spaces, because item alignment is lost if the right margin is justified (see "Constant Spaces and Character Translation" in topic 2.7.2.5.3).

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

LB-space is a value indicating the number of **nroff** blank lines (**troff** halves of vertical spaces) that are output by **.LB** itself. The value is 0 if the parameter is omitted.

There are three reasonable combinations of **LI-space** and **LB-space**.

Normally you will set **LI-space** to **1** and **LB-space** to **0**, yielding one blank line (a half vertical space, in **troff**) before each item in the list; such a list is usually terminated with a **.LE 1** to end the list with a following blank line (a half vertical space, in **troff**).

For a more compact list, you can set **LI-space** to **0** and **LB-space** to **1** and use **.LE 1** at the end of the list. The result is a list with one blank line (a half vertical space, in **troff**) before and after it.

You may want a very compact list; you can set both **LI-space** and **LB-space** to **0**, and use **.LE** to end the list. The result is a list without any separating vertical space.

The following "Advanced List Structures" in topic 2.7.8.7 shows how to build upon the supplied list macros to obtain other kinds of lists.

2.7.8.7 Advanced List Structures

If a large document requires complex list structures, it is useful to be able to define the appearance for each list level only once, instead of having to define it at the beginning of each list. This promotes consistent style for a large document. For example, you can define a generalized list-start macro in such a way that it performs a certain sequence depending on the list-nesting level in effect at the time the macro is called. Suppose that levels 1 through 5 of lists are to have the following marking and indentation style:

A. **Level 1**...

[1] **Level 2**...

Level 3...

a) **Level 4**...

+ **Level 5**...

The following code defines **.aL**, an alternate list macro that always begins a new list and determines the type of list according to the current list level. It creates list markings and indentation like that above.

The **mm** list macros use the number register **:g** to determine the current list level; its value is **0** if there is no currently active list. Each call to a list-start macro increments **:g** and each **.LE** call decrements it.

```
.de aL
'             \"Register g is used as a local temporary
'             \"to save :g before it is changed below.
.nr g \\n(:g
.if \\ng=0 .AL A  \" give me an A.
.if \\ng=1 .LB A \\n(Li 0 1 4  \" give me a [1]
.if \\ng=2 .BL  \" give me a bullet.
.if \\ng=3 .LB \\n(Li 0 2 2 a  \" give me an a)
.if \\ng=4 .ML +  \" give me a plus
..
```

You can use this macro with **.LI** and **.LE** instead of using **.AL**, **.RL**, **.BL**, **.LB**, or **.ML**. For example:

```
.aL
.LI
first line.
.aL
.LI
second line.
.LE
.LI
third line.
.LE
```

Output

A. first line.

[1] second line.

```
      B.  third line.
```

Note: The **.aL** macro and the **.bL** macro described following are not part of the memorandum macro package. They are given here as illustrations of functional macros that you might define.

There is another approach to lists that is similar to the **.H** mechanism. The list-start macro, as well as list-item and list-end macros, are all included in a single macro. That **.bL** (big list) macro requires a parameter 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       \" Take the level from the parameter, if any.
.el .nr g \\n(:g            \" If no parameter, use current level.
.if \\ng-\\n(:g>1 .)D  "**ILLEGAL SKIPPING OF LEVEL"
                             \" Increase only 1 level
.if \\ng>\\n(:g \{.aL \\ng-1  \" if g >:g, begin new list...
.nr g \\n(:g\}              \" and reset g to current level (.aL changes g)
.if \\n(:g>\\ng .LC \\ng    \" if :g >g, prune back to correct level
'                            \" If :g = g, stay within current list
.LI                          \" In all cases, get out an item
..
```

For **.bL** to work, you must change the previous definition of the **.aL** macro to obtain the value of the **g** register from its parameter rather than from **:g** register. Calling **.BL** without parameters causes it to stay at the current list level. The **mm .LC** (list clear) macro removes list descriptions until the level is less than or equal to that of its parameter. For example, the **.H** macro includes the call **.LC 0**. If you wish text to be resumed at the end of a list, simply insert the macro call **.LC 0** to clear out the lists completely. The example that follows illustrates the simplicity of this approach.

Input:

```
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.
```

Output:

```
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.
```

2.7.9 Footnotes

Like displays, footnotes are processed in an environment that is different from that of the body of the text. There are two macros that delimit the text of footnotes. There is also a register used to automatically number the footnotes and a macro that specifies the style of the footnote text.

Subtopics
2.7.9.1 Automatic Numbering of Footnotes
2.7.9.2 Delimiting Footnote Text
2.7.9.3 Format of Footnote Text
2.7.9.4 Spacing between Footnote Entries

2.7.9.1 Automatic Numbering of Footnotes

Footnotes are automatically numbered when you call the **F** (footnote numbering) string register immediately after the text to be footnoted, without any intervening spaces. You call the contents of the register **F** with the escape request ***F**. This places the next sequential footnote number (in a smaller point size, in **troff**) a half-line above the text to be footnoted.

2.7.9.2 Delimiting Footnote Text

There are a pair of macros that delimit the text of each footnote:

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

The **.FS** (footnote start) macro marks the beginning of the text of the footnote, and the **.FE** (footnote end) macro marks its end. The **label** on **.FS**, if present, is used to mark the footnote text; otherwise the value retrieved from the **F** number register by string register **F** is used as a string of numerals. You can intermix automatically-numbered and labeled footnotes but the footnote sequence number is not incremented when a footnote label is used rather than the **F** register value. To label a footnote, follow the text to be footnoted with **label** rather than ***F**. The text between **.FS** and **.FE** is processed in fill mode.

You can use labeled footnotes for information to be placed on the cover sheet (such as the title and abstract), but you cannot use automatically numbered footnotes for that purpose. Similarly, only labeled footnotes can be used with tables (see "Tables" in topic 2.7.4.11.3).

Note: You cannot nest footnotes within footnotes or displays. See the note in topic 2.7.4.11. You cannot use footnotes within a table, for instance, if you have enclosed a table within a display.

Examples:

Automatically numbered footnote.

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

Labelled footnote.

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

You should place the ***F** or **label** immediately following the word to be footnoted in the text. The text of the footnote enclosed within the **.FS/.Fe** pair should immediately follow on the next line. Thus, the first thing following the footnote string or label is the **.FS** macro on the next line. It is a good practice to append a constant space (see "Constant Spaces and Character Translation" in topic 2.7.2.5.3) to ***F** or **label** when they follow an end-of-sentence punctuation mark, to assure good sentence separation. The footnote marker will otherwise fill much of the space, making the sentences seem crowded together.

2.7.9.3 Format of Footnote Text

Within a footnote text, you can control the 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. You can call the **.FD** (footnote default) macro to select an appropriate style.

```
.FD [parm] [1]
```

The first parameter is a number from the left column of the following table that sets parameters for hyphenation, adjustment, text indentation, and label justification:

The second parameter, a numeral 1, resets automatic footnote numbering to 1 at each level 1 heading when you supply it. As an example, the input line:

```
.FD "" 1
```

maintains the default formatting style and causes footnotes to be numbered beginning at number 1 after each level 1 heading.

Table 7-5. Footnote Default Macro Parameters: The .FD macro

Value	Hyphenate	Adjust	Text Indent	Label Justification
0	.nh	.ad	Text indent	Label left justified
1	.hy	.ad	Text indent	Label left justified
2	.nh	.na	Text indent	Label left justified
3	.hy	.na	Text indent	Label left justified
4	.nh	.ad	No text indent	Label left justified
5	.hy	.ad	No text indent	Label left justified
6	.nh	.na	No text indent	Label left justified
7	.hy	.na	No text indent	Label left justified
8	.nh	.ad	Text indent	Label right justified
9	.hy	.ad	Text indent	Label right justified

10	.nh	.na	Text indent	Label right justified
11	.hy	.na	Text indent	Label right justified

A brief explanation of the second and third column typesetter request values (further information is available at the parenthesized references):

.ad Adjust margin. The default value of **.ad** request is to adjust both margins. When fill mode is off, adjustment is deferred. Other available settings are left adjust only, right adjust only, and center each line.

.hy Automatic hyphenation. Automatic hyphenation can be on or off, and **.hy** request has parameters to control last-line handling and permissible points for word splits.

.na No adjustment of margin. The right margin is ragged. Line fill takes place if fill mode is on.

.nh No hyphen (see "Hyphenation" in topic 2.7.2.5.4). Controls whether automatic hyphenation is on or off.

If the first parameter to **.FD** is out of range, the effect is as if **.FD 0** were specified.

If the first parameter is omitted or null, the effect is equivalent t **.FD 10** in **nroff** and to **.FD 0** in **troff**; these are the respective initial values.

For long footnotes that continue onto a following page, it is possible that if you are permitting hyphenation, the last line of the footnote on the current page is hyphenated. Except for this case (which you can control by specifying any even number for the first parameter to **.FD**, giving you no hyphenation), hyphenation across pages is inhibited by the **mm** command.

Footnotes are separated from the body of the text by a short rule line. Footnotes that continue to the next page are separated from the body of the text by a full-width rule. In **troff**, footnotes are set in type that is two points smaller than the point size used in the body of the text. In **nroff** the appearance of footnotes and footnote markers can be printer-dependent. This can also apply to footnote separation (see following).

2.7.9.4 Spacing between Footnote Entries

Normally, one **nroff** blank line (or a three-point vertical space, in **troff**) separates the footnotes when more than one occurs on a page. To change this spacing, set the **Fs** (footnote spacing) register to the desired value. For example:

```
.nr Fs 2
```

causes two blank lines (a six-point vertical space, in **troff**) to occur between footnotes.

2.7.10 References

There are several memorandum macro features that you can use for creating references. There are two macros that delimit the text of references: a register used to automatically number the references and a macro to produce reference pages within the document.

Subtopics
2.7.10.1 Automatic Reference Numbering
2.7.10.2 Delimiting References
2.7.10.3 Reference Page

2.7.10.1 Automatic Reference Numbering

You can cause automatic reference numbering by calling the contents of string register **Rf** immediately after the text to be referenced:

```
  Text to be referenced.\*(Rf
```

This calls a string that assigns and places the next sequential reference number (for **troff**, in a smaller point size) enclosed in brackets a half line above the text to be referenced. The reference count is accessed and incremented in the **Rf** (reference) number register. **mm** actually uses the **:R**, (reference print) number register to print the reference number. You can change the **:R** value or format, thereby changing the reference mark style without affecting the total count of references in **Rf**.

2.7.10.2 Delimiting References

The **.RS** (reference start) macro and the **.RF** (reference finish) macro delimit the text of each reference:

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

The **name** is the two digit name for a string that is assigned to the current reference number. (See "User-Definable Names" in topic 2.7.11.3 for naming conventions.) You can call it later in the document using an escape sequence of the form ***(aA** to reference text that must be labeled with a prior reference number. The string register name **aA** must be previously given as the parameter to a **.RS** macro. The reference string is printed enclosed in brackets, a half-line above the text to be referenced. Once you have defined a reference string register, a **.RS/.RF** pair is not needed for subsequent references to the same source. Instead, you can call the string register to repeat the reference string.

2.7.10.3 Reference Page

Memorandum and publication style documents automatically generate a reference page, entitled by default **References**, at the end of the document (before the table of contents and cover sheet are generated) and list it in the Table of Contents. This page contains all the reference items, each text string that is enclosed within **.RS/.RF** pairs. Reference items are separated by a half-line space unless the **Ls** (line space) register is set to 0 to suppress this spacing. You can change the reference page title by defining the **Rp** (reference page) string register using the define string macro:

```
.ds Rp "New Title"
```

You can use the **.RP** (reference page) macro to produce reference pages anywhere else within a document (for instance, after each major section). You do not need to call **.RP** to produce a separate reference page with default spacings at the end of the document; two **.RP** parameters let you control resetting of reference numbering and page skipping.

```
.RP [parm1] [parm2]
```

Table 7-6. Reference Page Macro, Parameter Values

Value	parm1 Meaning	parm2 Meaning
0	Resets reference counter (default).	Puts references on a separate page (default).
1	Does not reset reference counter.	Does not cause a following **.SK**.
2		Does not cause a preceding **.SK**.
3		Does not cause an **.SK** before or after reference list.

If an **.SK** is not issued by **.RP** (you specified **3** for the second parameter), a single blank line separates the list of references from the preceding text. To produce references at the end of each major section:

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

This is interpreted by the formatter as:

> Skip three lines
> Give references; do not reset the reference counter; do not cause preceding page break (but page break after the references).
> Start the numbered section **Next Section.**

2.7.11 Customizing Macros

You can write new macros and customize existing macro functions. This section describes the form of existing macros, requests, string registers, and number registers.

Note: All request, macro, and string register names are kept by the formatters in a single internal table. There must be no duplication among such names. Number register names are kept in a separate table.

The following conventions are used in following text to describe names:

8	Digit
m	Lowercase letter
M	Uppercase letter
x	Any alphanumeric character (**8**, **m**, or **M**).
s	Special character (any nonalphanumeric character).

All other characters used are literals; they stand for themselves and are highlighted in bold face in this text.

Subtopics

2.7.11.1 Defined Requests, Macros, and String Registers

Formatter Requests
: Formatter requests are of the form **mm** or **m8**. **c2** is the only formatter request of the form **m8**.

Memorandum macros
: Memorandum macros and memorandum macro string register names are of the form **M**, **MM**, and **Mm**. For example, the macros **P** and **HU** and the string registers **F**, **BU**, and **Lt**. There are three macros that are exceptions: **1C**, **2C**, and **nP**. There are seven defined accent characters of the form **s** (See "Character Accents" in topic 2.7.5.1). There are macros used internally by the **mm** command, with the names **)x**, **}x**, **]x**, **>x**, and **?x** that you cannot use or modify.

Constant Width Macros
: **.CD**, **.CN** **.CP**, **.CW**, and **.PC** are macros used by the **cw** command.

Equation Preprocessor
: The **eqn** and **neqn** commands use number registers and string registers of the form **88**. Two digit number registers and string registers are reserved for their use, and the use of **tbl**.

Table Preprocessor
: The **tbl** command uses these macros: **T&**, **T#**, and **TW**. It reserves all string register names of the form **m-**, **m+**, **mor**, **88**, **8m**, **^m**, **#m**, and **#s**. These string registers are of the general form **ms**, **8m**, **ss**, and **sm**.

Viewgraph macros
: The viewgraph macros are of the form **Mm**, **MM**, and **M**. Most two-character macros are uppercase equivalents of **troff** requests. **mvt** command uses all two-character string registers beginning with a right parenthesis **)** or a right square bracket **]**.

2.7.11.2 Defined Number Registers

Formatter
: Formatter registers are of the form: **mm** or **.x**, but there are two exceptions: **%** and **.$**.

Memorandum macros
: Memorandum macro registers are of the form **M8** and **Mm** (for example, the registers **H1** and **Fg**) and **M**. You can set **M** registers on the command line (see "Setting Number Registers from the Command Line" in topic 2.7.2.4.5). **mm** uses these registers internally: **:x**, **;x**, **#x**, **?x**, and **!x**.

Constant Width Preprocessor
: The **cw** command uses the **cE** and **cW** number registers.

2.7.11.3 User-Definable Names

Since many registers, macros, requests, and string registers are already used or reserved, you need to follow some simple rules about selecting labels for your own use. The following naming conventions are recommended.

Number Registers

m Use names that consist of a single lowercase letter for number registers. Avoid using single-digit labels and single special character labels; they are used by various preprocessors or macro packages.

String registers

ms Use labels that consist of a lowercase letter followed by a special character for string registers, for example: **a[**.

Macros

mM Use labels that consist of a lowercase letter followed by an uppercase letter for macros. (Remember that the label **.nP** is already used.)

Number-handling macros.

m8 Use labels that consist of a lowercase letter followed by a number for macros or string registers used for numeric values. (Remember that **c2** is already used.)

Number Registers

m Single-character lowercase letters are reserved for your use for number registers. You can set values of these registers from the command line (see "Setting Number Registers from the Command Line" in topic 2.7.2.4.5). Single-digit register names are used by **eqn/neqn**.

2.7.11.4 Extending Macros and Creating Macros

The following are examples of ways to extend existing macros to achieve new purposes. The technique involved is to use existing macros and registers. You must be familiar with programming concepts to modify or assemble macros for new purposes or to write new macros.

Note: *If* you are an experienced programmer with knowledge of the AIX PS/2 Operating System, you may wish to examine macro package files contained in directory **/usr/lib/tmac**. As existing macros are often interdependent, altering macros or creating new macros that interact with existing ones can have unforeseen effects. Where possible, you should take advantage of the existing flexible macro packages and the various file processing commands. Many desirable features are already available directly or by extending macros as illustrated in the following sections.

Subtopics
2.7.11.4.1 Appendix Headings
2.7.11.4.2 Tabulated Hanging Indent

2.7.11.4.1 Appendix Headings

You can generate and number appendix headings by extending the **.PH** (page header) macro (see "Page Header" in topic 2.7.4.3.1).

```
.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 you can save for the table of contents if you desire. If you also want appendix titles to be centered, you must set the **Hc** (heading centering) register to **1** (see "Altering Appearance of Headings" in topic 2.7.6.2).

2.7.11.4.2 Tabulated Hanging Indent

The following example illustrates the use of the hanging-indent feature of the variable-items lists (see "Variable-Item List" in topic 2.7.8.2.5).

First, you build a user-defined macro to accept four parameters tha make up the **mark**.

In the output, each parameter is to be separated from the previous on by a tab; tab settings are defined later.

Since the first parameter can begin with a period or apostrophe, th **\&** is used so the formatter does not interpret such a line as a formatter request or macro call. (The two-character sequence **\&** is understood by the formatters to be a zero-width space. It causes no output characters, 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 macro and an example of its use are:

```
.de aX
.LI
\&\\$1\t\\$2\t\\$3\t\\$4\t\c
..
.
.
.
.ta 9n 18n 27n 36n
.VL 36
.aX .nh off \- no
No hyphenation.
Automatic hyphenation is turned off.
Words containing hyphens
(such as mother-in-law) can still be split across lines.
.aX .hy on \- no
Hyphenate.
Automatic hyphenation is turned on.
.aX .hc\&ballot.c none none no
Hyphenation indicator character is set to "c" or removed.
During text processing the indicator is suppressed
and will not appear in the output.
Prefixing the indicator to a word has the effect
of preventing hyphenation of that word.
.LE
```

In the above example, the &ballot. symbol represents a blank space character, for clarity. The resulting output is:

.nh	off	--	No	No hyphenation. Automatic hyphenation is turned off. Words containing hyphens (such as mother-in-law) can 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. Prefixing the indicator to a word has the effect of preventing hyphenation of that word.

2.7.12 Debugging and Diagnostics

When a formatter (or **mm** command) discovers an error, the following actions occur:

1. A break occurs.

2. To avoid confusion regarding the location of the error, the formatter output buffer (which can contain some text) is printed.

3. A short message is sent to the originating video display, giving:

   The name of the macro that found the error
   The type of error
   The approximate line number (in the current input file) of the last processed input line.

   All the memorandum macro and formatter error messages are explained under "mm Error Messages" in topic D.1.

4. Processing stops, unless the **D** (debug) register has a positive value (see "Setting Number Registers from the Command Line" in topic 2.7.2.4.5). In the latter case, **mm** attempts to continue processing, even though the output (if any) is probably garbled.

   If an output filter, such as **hp** is being used to post-process **nroff** output, an error message can be intermixed and garbled with text held in that filter output buffer.
   If any of **cw**, **eqn**, **neqn**, or **tbl** are being used and if the **-olist** option of the formatter causes the last page of the document not to be printed, a harmless **BROKEN PIPE** message can result.

Output can be lost, usually because of an unclosed diversion (probably from a missing **.DE** or **.FE** in input text). (See "Displays" in topic 2.7.4.11, and "Footnotes" in topic 2.7.9.) Fortunately, **mm** checks to assure that macros using diversions do not permit illegal nesting. If any message is issued about a missing **.DE** or **.FE**, search backwards from the point that processing stopped, looking for a **.DF**, **.DS**, or **.FS** that is missing a closing **.FE**.

The following system command 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 its line number in that file. You can use this listing to check for illegal nesting or omission of macros.

```
grep -n "(^)\.[EDFRT][EFNGS]" files ...
```

2.7.13 Sample of Letter Style

Note: The **nroff** and **troff** output that corresponds to the following input text is shown on following pages.

```
.ND "June 29, 1984"
.TL
Course Description
.AF "ACME Metals"
.AU "D. W. Stevenson" DWS PY 9876 5432 1X-123
.MT 0
.DS
J. M. Jones:
.DE
.P
Please use the following description for Course No. 2275,*
"Document Preparation on the AIX PS/2 Operating System":
.FS *
This course is a prerequisite for Course No. 2277.
.FE
.P
The course is intended people who intend
to use the AIX PS/2 Operating System for preparing documentation.
The course will cover such topics as:
.VL 18
.LI "Text editing:"
how to enter text so that subsequent revisions are easier to make;
how to use the editing system to add, delete, and move lines of text;
how to make corrections.

.LI "Text processing:"
basic concepts; use of general-purpose formatting packages.
.LI "Other facilities:"
additional capabilities useful to the typist such as the
.I "spell, diff,"
and
.I grep
commands, and a desk-calculator package.
.LE
.SG jrm
.NS
S. P. Lename
.NE
```

ACME Metals

subject: Course Description

date: June 29, 1984

from: D. W. Stevenson
PY 9876
1X-123 x5432

J. M. Jones:

Please use the following description for Course No. 2275,(*) "Document Preparation on the AIX PS/2 Operating System":

The course is intended for clerks, typists, and others who intend to use

the AIX PS/2 Operating System for preparing documentation. The course will cover such topics as:

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

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

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

PY-0987-DWS-jrm **D. W. Stevenson**

Copy to
S. P. Lename

*** This course is a prerequisite for Course No. 2277.** (*)

ACME Metals

subject: **Course Description** date: **June 29, 1984**

from: **D. W. Stevenson**
PY 9876
5432 IX-123

J. M. Jones:

Please use the following description for Course No. 2275, "Document Preparation on the AIX PS/2 Operating System":

The course is intended for clerks, typists, and others who intend to use the AIX PS/2 Operating System for preparing documentation. The course will cover such topics as:

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

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

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

PY-0987-DWS-jrm **D. W. Stevenson**

Copy to
S. P. Lename

(*) This course is a prerequisite for Course No. 2277.

2.8 Chapter 8. Using nroff and troff

Subtopics

2.8.1 About This Chapter

The **nroff** and **troff** formatters prepare text for printing on a printer or phototypesetter, respectively. The **troff** utility is a text formatting program designed to prepare text for phototypesetting on a Graphic Systems Inc. or Wang Laboratories C/A/T phototypesetter. The **troff** formatter provides the ability to create a high-quality text appearance. Additional utilities prepare **troff** output for viewing on a special workstation screen or suitable printer or for printing on other typesetting equipment. The **nroff** formatter is a parallel utility to **troff**, created to provide text formatting for readily available printers and workstations. The **nroff** formatter creates output generally suitable to whatever device you favor.

Note: Most **nroff** requests are understood by **troff** and visa versa. A chief difference is in machine unit measures (see "Formatter Resolution, Scales, and Machine Units" in topic 2.8.2.4). Where documents are to be used for generating text using both formatters, you should avoid specifying dimensions in machine units. Another difference between **nroff** and **troff** is that **nroff** generally assumes that a printer only has a single font and type size available, at a fixed pitch; **troff** assumes you have up to four fonts available, in 15 type sizes, and offers precision placement of text. You will otherwise find this guide to be generally applicable to both **nroff** and **troff**. Where **troff** and **nroff** features differ, the text will indicate it. The term ***formatter*** is generally used where both **nroff** and **troff** are meant.

You cannot use the **nroff** and **troff** programs to format Japanese text as these programs do not support multibyte characters.

The **troff** formatter drives a typesetter that normally uses four fonts, most often Roman, italic, and bold character sets and a special mathematics character set having a full Greek alphabet, special formula-constructing characters, and mathematical symbols. (Each character set consists of 102 symbols.) Other fonts can be mounted in the phototypesetter; there are hundreds of fonts available on typesetters that **troff** can drive. A powerful range of formatting tools are under your control, including proportional spacing, automatic hyphenation, page titling and numbering, footnoting, and so on. You can cause characters to be printed in a variety of sizes and placed anywhere on a page.

For complex tasks, the formatter provides macros (including definable macros), arithmetic variables and operations, and case testing. In most cases, you will prefer to prepare text using a macro package, such as the memorandum macros and call preprocessor filters, such as **eqn** or **tbl** to prepare text for formatter processing. The **troff** and **nroff** formatters are powerful and flexible, but their complexity makes them difficult to use directly. You must give formatter operations in great detail to use them effectively, and few users need to know the formatter that well. You will find it useful to become familiar first with the macros and filter utilities, learning the finer details of the formatter as you need to use some special features:

For producing straight text (which may contain mathematics or tables) you can use a number of ***macro packages*** that define formatting rules and operations for specific styles of documents, and reduce the need to directly access the formatter.

- In particular, the memorandum macros provide a wide range of facilities for document preparation from simple memorandums to books.

- The Viewgraph Macro package is used for preparing foils for presentations on viewgraph and slide media with the **troff** formatter.

There are other macro packages designed for special purposes. You will find these packages easier to use than using a formatter directly once you get beyond very trivial operations; you should always consider them first. The macro packages that are supplied with this system are stored in the directory **/usr/lib/tmac**; you can create other macro packages or acquire commercially available packages.

eqn (see Chapter 9, "Formatting Mathematics") is a filter for formatting mathematics; you do not need to know about **nroff** or **troff** to typeset mathematics. **eqn** can be called from **troff** or from the **mm** command; **neqn** is the parallel utility that preprocesses text for **nroff**.

tbl (see Chapter 10, "Formatting Tables") provides the same convenience for producing tables and works with either formatter.

cw provides a preprocessing filter in a constant-width font that is compatible with **tbl**; you cannot use **cw** with **eqn** because **cw** uses a nonstandard set of characters that makes it unable to map some mathematical characters. **cw** is particularly useful to replicate the appearance of a video display or line printer output. You can use **cw** to print out a program listing, for instance. Processed **cw** output is piped to a formatter or other preprocessor input. **cw** text is generally imbedded in a **troff** display.

col postprocesses an **nroff** file to provide interpretation of some character calls and to buffer text for output to printers that are not capable of backscrolling and half-linefeed motions.

Occasionally, you may want to do something that is not provided for by a macro package. In that case, a few small additions to formatter requests or defining a couple of new macros is usually your best solution. You will not have to use a formatter directly very often, as most formatting features are already handled by existing macro packages. If, however, you are performing extensive text formatting in a variety of styles, you will need to know **troff** (or **nroff**) well. This chapter provides you with information you need in order to use a formatter effectively.

2.8.2 Typesetter Command Line

The general form of invoking a formatter at the command level is:

nroff [**flags**] [**files**]

or

troff [**flags**] [**files**]

where **flags** represents any of a number of option parameters and **files** represents the list of files containing the document to be formatted. A parameter consisting of a single minus sign (**-**) is taken to be a file name corresponding to the standard input. If file names are not given, input is taken from the standard output. The flags, which can appear in any order so long as they appear before the files, are:

Flag	Effect of the Flag
-olist	Prints only pages with page numbers that appear in **list**, which consists of comma-separated numbers and number ranges. A number range has the form **num-num2** and means pages **num** through **num2**; an initial -**num** means from the beginning to page **num**; and a final **num**- means from page **num** to the end.
-nnum	Numbers first page **num**. The revised numbering scheme is used to choose pages if the **-o** flag is present.
-snum	Stops every **num** pages. **nroff** halts prior to every **num** pages (default **num**=1) to allow paper loading or changing, and resumes upon receipt of a new-line character from the workstation. **troff** stops the phototypesetter every **num** pages, produces a trailer to allow the typesetter operator to load machine paper, and resumes after the phototypesetter START button is pressed.
-mname	Reads the macro file**/usr/lib/tmac/tmac.name** to input **files**.
-ranum	Sets the number register named **a** to **num**.
-i	Reads standard input after the input files are exhausted.
-q	Calls the simultaneous input/output mode of the **rd** request.
-z	Prints messages generated by **.tm** requests.

Subtopics

2.8.2.1 nroff Only

Flag	**Effect of the Flag**
-Tname	Specifies the name of the output terminal type. The value **37** specifies the TELETYPE Model 37, which is the default.
-e	Produces equally spaced words and adjusted lines, using full-terminal resolution. (The resolution of the IBM Personal Computer Graphics Printer, for instance, is one character space.)

2.8.2.2 troff Only

Flag	Effect of the Flag
-t	Directs output to the standard output instead of the phototypesetter.
-f	Inhibits feeding out paper and stopping phototypesetter at the end of the run.
-w	Waits until phototypesetter is available if the typesetter is currently in use.
-b	Causes **troff** to report whether the phototypesetter is busy or available. Text processing is not done.
-a	Sends a printable (ASCII) approximation of the results to the standard output.
-pnum	Prints all characters in point size **num** while retaining all prescribed spacings and motions, reducing phototypesetter elapsed time.

Each flag is called as a separate parameter, for example:

```
nroff -o4,8-10 -Tlp -mabc file1 file2
```

This example requests formatting of pages **4**, **8**, **9**, and **10** of the document contained in files named **file1** and **file2** and reads the macro package **abc** before reading the files.

(The default output is acceptable for printing on the IBM Personal Computer Graphics Printer if postprocessed with **col -x**.)

2.8.2.3 Formatter Input

Input consists of text that is to be output, interspersed with control sequences (control line requests and escape requests), which control processing.

Control lines begin with a ***control character***, normally a period or an apostrophe (') followed by a one- or two-character name that specifies a basic request or the substitution of a user-defined macro. The control character ' suppresses the formatter break function (the forced output of a partially filled line) that is caused by some requests. You can separate the control character from the request or macro name by white space (spaces and tabs). The formatter ignores control lines with unrecognized names.

You can introduce escape requests anywhere in the input by means of an ***escape character***, normally the backslash \ character. For example, the escape request **\nr** causes the contents of the number register **r** to be read. See Table 8-2 in topic 2.8.9.1 for more information about the effect of escape sequences on number registers.

2.8.2.4 Formatter Resolution, Scales, and Machine Units

The **troff** formatter internally uses 432 units per inch and has a horizontal resolution of 1/432 inch and a vertical resolution of 1/144 inch. **nroff** internally uses 240 units/inch, corresponding to the least common multiple of the horizontal and vertical resolutions of some typewriter-like output devices. **troff** rounds horizontal and vertical numerical parameter input to its internal horizontal and vertical resolution. **nroff** similarly rounds numerical input to the actual resolution of the output device indicated by the **-T** flag.

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

Table 8-1. Formatter Basic Unit Conversions

Indicator	Translates To	troff Units	nroff Units
c	Centimeter	170.1 (approx.)	94.4 (approx.)
i	Inch	432	240
m	Em (type size point value)	6 ¦ number of type size points	**N** (1)
n	En (half an em)	3 ¦ number of type size points	**N** (1)
P	Pica (1/6 inch)	72	40
p	Point (1/72 inch approx.)	6	3.33 (approx.)
u	Basic unit	1	1
v	Vee (2)	**V** (2)	**V** (2)
none	Default		

The default scaling is:

Default Situations

m For the horizontally oriented requests: **.in**, **.ta**, **.ti**, **.ll**, **.lt**, **.mc**, **.po**, **\h**, and **\l**.

v For the vertically oriented requests: **.ch**, **.dt**, **.ne**, **.pl**, **.rt**, **.sp**, **.sv**, **.wh**, **\L**, **\v**, and **\x**.

p For the **.vs** request.

u For the requests: **.ie**, **.if**, and **.nr**.

All other requests ignore scale indicators.

When a macro takes a value from a number register and uses it, a scaled number value is converted to basic units. If you are designing a macro that gets a number register value, you may need to add the **u** (basic unit) scale indicator to prevent inappropriate default scaling for some requests. The register number value is specified in decimal-fraction form, but the parameter finally stored is rounded to an integer number of basic units.

You can prefix an absolute position indicator, **|**, to a number **locval** to set the distance to the vertical or horizontal position **locval**. For vertically oriented requests, **|locval** becomes the distance in basic units from the current vertical place on the page or in a diversion to the vertical place **locval**. For all other requests, **|** becomes the distance from the current horizontal place on the input line to the horizontal place **locval**. For example:

```
.sp |3.2c
```

spaces 3.2 centimeters from the top of the page (in the required direction).

(1) In **nroff**, an em and an en are both **N** width, the nominal character width value, equal to the value of the pitch (1/**x** inch). Actual character width varies, except when a constant-width character set is used.

(2) A vee (**V**) is equal to current line spacing, expressed in basic units. The **troff** point value of vee is variable, initially type size + 2; the **nroff** point value of vee is fixed and output-device dependent.

2.8.2.5 Numerical Expressions

The formatter accepts the following (as well as whole and decimal fraction numbers) wherever it expects numeric input:

Arithmetic operators

+
-
/
*****
% (mod)

Logical operator

<
>
<=
=>
= (or **==**)
& (and)
: (or)

Parenthese

(
)

Except where controlled by parentheses, expressions are evaluated 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 respectively. If a value has a default scale, you must attach the desired scale indicator 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
```

sets the line length to half the sum of 4.25 inches + 2 picas + 30 points.

2.8.2.6 Request Number-Handling

Generally, the formatter ignores an unreasonably long numerical input or truncates it to a reasonable value. Most requests expect to set parameters to non-negative values; exceptions are **.ch**, **.if**, **.nr**, **.sp**, and **.wh**. The following requests restore the previous parameter value if you do not provide a parameter with them: **.ft**, **.in**, **.ll**, **.ls**, **.lt**, **.po**, **.ps**, and **.vs**.

2.8.3 Fonts

The **troff** formatter and the C/A/T typesetter allow four different fonts to be used at any one time. Normally three fonts (Roman, italic, and bold) and one collection of special characters are permanently mounted:

Font	Example
Roman	abcdefghijklmnopqrstuvwxyz ABCDEFGHIJKLMNOPQRSTUVWXYZ 0123456789
Italic	**abcdefghijklmnopqrstuvwxyz ABCDEFGHIJKLMNOPQRSTUVWXYZ 0123456789**
Bold	**abcdefghijklmnopqrstuvwxyz ABCDEFGHIJKLMNOPQRSTUVWXYZ 0123456789**

troff was designed with the assumption that Times Roman, Times Italic, and Times Roman Bold fonts are the fonts that are mounted on the typesetter.

Subtopics
2.8.3.1 Font Selection and Control
2.8.3.2 Emphasized Print
2.8.3.3 Type Size

2.8.3.1 Font Selection and Control

The **troff .ft** (font) request selects the font to use:

```
.ft [fontchar]
```

where:

fontchar	Effect of fontchar
.ft B	Sets the selected font to bold.
.ft I	Sets the selected font to italic.
.ft P	Sets the selected font to the last previously selected font. The **P** is the default if a parameter **fontchar** is not given.
.ft R	Sets the selected font to Roman. **troff** formats for Roman font unless told otherwise.
.ft S	Sets the selected font to a special font.

You can change fonts in other ways. For instance, the underline request, **.ul**, is interpreted by **troff** as a request to italicize. **.ul** causes the next input line to print in italics. **.ul 5** causes the next five input lines to be printed in italics. The sixth line and those following would be printed in whatever font was in effect before the **.ul** request.

You can also change fonts within a line or word with the escape request **\f**. (A backslash is used to prefix all escape requests.) For example:

```
\fBBold\fIface\fR text.
```

produces the following:

Boldface text.

To do this so the previous font is left undisturbed (a good practice), insert extra **\fP** requests, like this:

```
\fBBold\fP\fIface\fP\fR text.\fP
```

The formatter only remembers the immediately previous font, so you should restore the previous font after each change. If you switch from Roman to bold and then switch directly to italics, selecting previous font (**.fP** or **.ft**) returns you to bold rather than to Roman. (The **.ps.** and **.vs** requests also react the same way when used without a parameter.) The previous font value is stored in the **.f** (font) register.

Note: The **nroff** formatter understands font control requests, but it cannot always use them. By default, it underlines when the selected font is italic.

troff assumes specific fonts are on a given physical device (typesetter mount position). The RIBS mnemonic may help you remember what you are asking for with an **.ft** request, since R, I, B, and S type selections correspond to physical device positions 1, 2, 3, and 4 respectively. You will need to remember this if you mount some special type on a physical device. For instance, you may want to print a document in Helvetica font, without giving up Roman font:

```
.fp 3 H
```

This instructs the typesetter to mount Helvetica font on position 3. (Appropriate **.fp** requests should appear at the beginning of your document if you do not use the standard fonts.) Now when **troff** sees a request for an italic type (**.ft I**), it causes the typesetter to use the Helvetica typeface that is loaded on physical device 3, and not italic.

You can select a font by using font numbers instead of names; for example, **\f3** and **.ft 3** and **.ft B** all tell the typesetter to use whatever font is mounted at position three. If you will be printing a document in different fonts at different times, you may prefer to select fonts by device number.

2.8.3.2 Emphasized Print

There is a way to create artificial bold fonts by ***overstriking*** letters, printing each one twice with a slight offset; the effect is known as ***emphasized print***. This is useful if you need to put some particular font on device three or if you want to put several different types in bold face in the same document. The advantage is that you can match any selected type in a document with a corresponding boldface type; the disadvantage is that the artificial boldface type does not look as crisp or proportionally correct as a true boldface type. You can also create a very heavy print by emphasizing a bold type face.

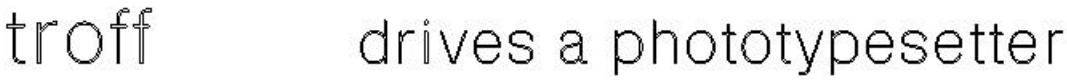
2.8.3.3 Type Size

The **troff** request **.ps** sets point size. **nroff** ignores this request. One point is 1/72 inch, so 6-point characters are at most 1/12 inch high, and 36-point characters are no taller than a half inch. There are 15 point sizes that troff normally supports: 6, 7, 8, 9, 10, 11, 12, 14, 16, 18, 20, 22, 24, 28, and 36.

If you give a number after **.ps** that is not a recognized size, it is rounded up to the next valid value, to a maximum of 36 for most fonts. If you give **.ps** with no following value, **troff** returns to the previous set size. **troff** assumes an initial point size of 10. Most documents are set in 9 to 12 point type, with 10 and 11 point type being most common. You can change the point size in the middle of a line or even a word with the escape request **\s**. **\s** should be followed by a legal point size; **\s0** causes the size to revert to its previous value. For example:

```
\s8troff drives a \s6phototypesetter
```

results in:



Note: A compound request like **\s1011** is interpreted by **troff** as a size 10 character followed by size 11 when the selected type sizes are recognized by **troff** but not otherwise. Be cautious with compound constructions.

Relative size changes are very useful. A request of the form:

```
\s-2... (intervening text) ...\s+2
```

temporarily decreases whatever the current type size is by 2 points, then restores it. Relative size changes have the advantage because size difference is independent of the starting size of the document. Such differential requests are relevant even when the document on the whole is

reformatted to a different type size, such as changing the base type size from 10 point to 12 point. A relative change request is restricted to a single digit value; therefore, a 9 point differential is the largest you can set.

The **.ps** request given without a parameter causes the point size to revert to the previous setting.

The current point size is stored in the **.s** register.

2.8.4 Special Characters

Special characters have four-character names beginning with **\(**, and they can be inserted anywhere on an input line. For example:

```
\(14 + \(12 = \(34
```

produces:

1/4 + 1/2 = 3/4.

In particular, Greek letters are of the form **\(*x** or **\(xx** where **x** is a capitalized or lowercase letter in the modern alphabet that represents a Greek character. For example:

```
\(*S(\(*a\(mu\(*b) \(-> \(if
```

is interpreted by **troff** as follows:

troff Input	**troff Output**
\(*S	&Sigma.
(	(
\(*a	&alpha.
\(mu	¦
\(*b	ß
)	)
\(->	
\(if	&infinity.

The result looks like this:

&Sigma.(&alpha.¦ß) &infinity.

Using the **eqn** preprocessor (see Chapter 9, "Formatting Mathematics" in topic 2.9), you can achieve the same effect with the input:

```
SIGMA ( alpha times beta ) -> inf
```

Note: Each four-character escape request is a single character as far as the formatter is concerned.

Subtopics
2.8.4.1 Overstriking Requests
2.8.4.2 Zero-Width Characters
2.8.4.3 Equation Construction Characters and Vertical Assembly
2.8.4.4 Line Drawing and Underlining

2.8.4.1 Overstriking Requests

The formatter constructs some symbols by overstriking characters. You can automatically cause centered overstriking of up to nine characters at a time using the formatter **\o** (overstrike) escape request, for example:

```
\o'string'
```

The characters in **string** are overprinted with centers aligned; the total character width is that of the widest character in **string**. The string of characters should not contain local motion.

The character following the **\o** request (in this case ') is taken to be the delimiter. The character ' is used as the delimiter in examples throughout this book.

2.8.4.2 Zero-Width Characters

You can produce left-aligned overstruck combinations using the **\z** (zero-width character) escape request:

```
\z char
```

\z marks the current location, outputs the following **char**, and returns to the marked location.

2.8.4.3 Equation Construction Characters and Vertical Assembly

The **troff** formatter uses the special mathematical font to let you assemble large formulae, including large square brackets, braces, and continuous vertical and horizontal lines, as well as including any defined characters of any available character set. The **nroff** formatter also recognizes the requests for the special mathematical font and creates output that simulates **troff** output; the resulting appearance is output device-dependent.

The special mathematical font contains a number of bracket construction pieces that you can combine into various large output constructions. These pieces are designed to let you construct complex mathematic formulae. You can assemble characters vertically with the **\b** (bracket) escape request:

```
\b'string'
```

The characters in **string** are built up vertically, with the first character on top and the last at the bottom. The characters are vertically spaced by 1 em spacing and the total ***pile*** is centered (in **troff**) half an em above the current baseline (half a line above, in **nroff**). For example:

```
\b'\(lc\(lf'E\|\b'\(rc\(rf'\x'-0.5m'\x'0.5m'
```

produces:

```
left lbracket 'E' right rbracket
```

2.8.4.4 Line Drawing and Underlining

The formatter provides a facility for drawing horizontal and vertical lines of arbitrary length with arbitrary characters. **\l'1i'** draws a line 1 inch long, like this: ------. The length can be followed by the character to use if the _ character is inappropriate; **\l'0.5i.'** draws a half-inch line of dots: ..... The construction **\L** is entirely analogous, except that it draws a vertical line instead of horizontal. For example, **\L'3v'** produces:

|
|
|

The **\l** (line draw) escape request assembles a horizontal string of a specified character (**char**) by repeating the character the number of times necessary to fill the specified distance (**hval**):

\l'**valchar**'

The character **char** is repeated from the current location to the right for a distance **val**. If **char** looks like a continuation of an expression for **num**, you can insulate it from **val** with a **\&** (zero-width space). If **char** is not specified, the baseline-rule character is used in **troff**; **nroff** uses an underscore character as an equivalent. If **val** is negative, a backward horizontal motion of **val** is made before drawing the string to the right. Any space resulting from **val**/(size of **char**) having a remainder is put at the beginning (left end) of the string. In the case of **troff** characters that are designed to be connected, such as the _ (baseline-rule), _ (underscore) and root-en ((-)), the remainder space is filled by overlapping.

If **val** is less than the width of **char**, a single **char** is centered a distance **val** right of current position. As an example, you can define a macro to underscore a string:

```
.de us
\\$1\I'|0\(ul'
..
```

which could be used so that :

```
.us "underlined words"
```

yields:

underlined words

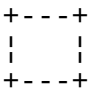
You can define a macro to draw a box around a string:

```
.de bx
\(br\|\\$1\|\(br\'|0\(rn'\'|0\(ul'
..
```

With the preceding definition, this request:

```
.bx " "
```

gives you:



The **\L** (vertical line) escape request draws a vertical string consisting of a character stacked vertically one em apart in **troff** (one line apart in **nroff**). In **troff**, the characters are overlapped, if necessary, to form a specified length of continuous line, just as are horizontal line draw characters. The request is made in this form:

\L'**valchar**

If you do not specify **char**, the default character is the ***box rule character*** **|**, specified by the escape request **\(br**. Another character suitable to produce vertical lines is the **|** called by the **\(bv** (bold vertical) escape request. The line is begun without any initial motion relative to the current base line. A positive **val** value specifies a line drawn downward. A negative **val** specifies a line drawn upward. After the line is drawn, no compensating motions are made; the new baseline is at the end of the drawn line. If you specify a **char** that is not a full em in height, there can be a space between the characters as drawn. There will be spaces between **nroff** characters in any case (unless a box rule character is part of the recognized character set) as a line is longer than 1 em vertically.

The **nroff** formatter automatically underlines characters in the underline font you can specify with the **.uf** (underline font) escape request. This is normally set for position 2, what would typically be Times Italic in **troff**. In addition to the **.ft** and **\ffont**, the underline font is selected by the **.ul** and **.cu** requests. Underlining is limited to an output device-dependent set of reasonable characters.

You can use horizontal and vertical line drawing requests in combination to produce large boxes. The zero-width box rule and the half em wide underscore were designed to form corners (**troff**) when using 1 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'or\\
nau-1'\1'\\n(.lu+1n\(ul'\L'-or\\nau+1'\1'or0u-.5n\(ul'   \"draw box
.fi
..
```

draws a box around some text that has its beginning vertical place saved in number register **a** using the request **.mk a**.

2.8.5 Line Filling, Adjusting, and Centering

Normally, the formatter collects words from input text lines and assembles them into a single output text line until some word does not fit (not enough room is left for it on the line). It then attempts to hyphenate the word in an effort to fill as much of the output line as possible with text. Then, it expands spaces between words to spread the line to the current line length minus any current indent.

The formatter assumes that an input text line ending with a period, question mark, or exclamation mark is the end of a sentence; the formatter adds an additional space character to output automatically during filling. Multiple interword space characters found in the input are retained, except for ***trailing spaces*** (spaces at the end of a line). Initial spaces cause a break and are retained for output.

When filling is in effect, a word with an imbedded or attached **\p** receives a break at the end of the word; the resulting output line is expanded (spread out) to fill the current line length. You can control or prevent filling, adjustment, and hyphenation.

You can cause a text input line that happens to begin with a control character to be treated as input text by prefacing it with the nonprinting **\&** (zero-width filler) escape request. Another way to keep lines that appear to be requests from being treated as such is to specify output translation of some convenient character into the control character using the **.tr** request.

Subtopics

2.8.5.1 Hyphenation and Fill

The formatter has an automatic hyphenation feature. You can select whether it is switched off or on; it is initially off. The **.hy** request sets hyphenation, and you can set several variables. You can specify an exception word list to be used and a ***hyphenation escape request*** that is imbedded in a word to specify desired hyphenation points. (The escape request does not appear in output.) You can prefix a word with the hyphenation escape request to cause automatic hyphenation to be suppressed for the word.

The formatter only considers words that consist of a central alphabetic string surrounded by (usually null) nonalphabetic strings as candidates for automatic
hyphenation.

Note: Words that are input containing hyphens (minuses), em-dashes (**\(em**), or hyphenation indicator characters are always subject to splitting after those characters, regardless of whether or not automatic hyphenation is selected.

Hyphenation is turned on or set with the **.hy** (hyphenation) request:

.hy [**num**]

Hyphenation is off when **num** equals zero; **num** can be **0** or 1, or 2, 4, 8, or a sum of a combination of numbers 2, 4, and 8. If **num** is greater than **0**, automatic hyphenation is on. If **num** is:

1, automatic hyphenation is turned on

2, last lines (lines that cause a trap, as when bottom-of-page i reached) are not hyphenated.

4, the last two characters of a word are not split off. (For example a hyphenation like "Bron-te" would not occur.)

6, the results are as if both 2 and 4 are set

8, the first two characters of a word are not split off. (Fo example, "ot-her". would not occur.)

10, the results are as if both 2 and 8 are set

12, the results are as if both 4 and 8 are set

14, the results are as if 2, 4, and 8 are set

A **.nh** (no hyphenation) request also turns hyphenation off and is the equivalent of the request **.hy 0**.

You can select the hyphenation token form using the **.hc** (hyphenation character) request:

.hc [**string**]

By default, **string** is **\%**.

The ***exception word list*** is a brief list of words that you can specify to be hyphenated in a particular way, or not at all. The total length of the

list cannot exceed 128 characters. The **.hw** (hyphenate words) request is of the form:

```
.hw [word] ...
```

Each **word** is a word including no hyphens if you wish it to always appear unhyphenated or including an ASCII hyphen (minus) character (not the hyphen escape request) at each acceptable hyphenation point in the word.

The formatter examines the list for **word** after each suffix stripping.

The **.fi** fill request sets fill on by storing a value of 1 (the default value) in the **.u** number register. A value of 0 in the **.u** register sets fill off. You can turn fill off with a **.nf** (no fill) request, which turns off both fill and adjust functions (see "Text Adjustment" in topic 2.8.5.2).

2.8.5.2 Text Adjustment

A ***word*** is any string of characters delimited by the space character or the beginning or end of the input line. Any adjacent pair of words that must be kept linked together (neither split across output lines nor spread apart in the adjustment process) can be tied together by separating them with the backslash-space (\ ) ***constant space character***. The adjusted word spacings are uniform in **troff**, and you can set the minimum interword spacing with the **.ss** request:

.ss **num**

The **num** is the minimum character space in 36ths of an em. The **nroff** formatter ignores this request; inter-word spacing is often not uniform because many printers do not support fractional spaces for line adjustment.

You can set line adjustment with the **.ad** (adjust) request. When fill mode is not on, adjustment is deferred until fill mode is turned on. You can use this request to set the type of adjustment performed:

.ad [**char**]

The **char** can be omitted, which means no change occurs in the type of adjustment. The **char** can be:

char	**Effect of char**
b	Adjusts both margins
c	Centers
l	Adjusts left margin only
n	Adjusts both margins
r	Adjusts right margin only.
num	A **num** value taken from the **.j** register to restore the previous adjustment mode.

You can turn adjustment off with the **.na** (no adjust) request.

Adjustment does not take place in the **troff** constant space mode. Constant space mode is selected with the **.cs** (constant space) request, which has no effect in **nroff**.

2.8.5.3 Interrupted Text

If line filling is off, you can force a line to be filled with additional text by using a **\c** request as the last entry on the line. This causes the formatter to consider the next line as a continuation of the previous line. Similarly, you can interrupt a line of filled text by interrupting a word (and line) with **\c**; the next encountered text is taken as a continuation of the interrupted word. If an intervening control line causes a break, any partial line is forced out along with any partial word.

2.8.5.4 Centered Text Lines

You can cause a line or lines to be centered using the **.ce** (centering) request.

```
.ce [num]
```

where **num** is a positive number or zero that indicates the number of following lines to center. The default **num** is 1, causing only the following line to be centered. (**.ce** causes a break for each centered input line.) A **num** of **0** causes the centering counter to be cleared, and following lines are not centered. If an input line is too long to center, it is left unadjusted.

2.8.6 Spacing

There are a number of requests that control horizontal and vertical spacing within a document.

Subtopics
2.8.6.1 Inserting Vertical Space
2.8.6.2 Reserving Block Space

2.8.6.1 Inserting Vertical Space

The **troff** request **.sp** is used to insert extra vertical space. If you follow the request with no value, **troff** assumes a value of one unit of vertical space. (Vertical space is initially set by the formatter as type size plus 2 points and is a value stored in the **.v** register.) Usually you will want to insert vertical space in lines (or decimal fractions of lines), because the inserted space will then remain in relation to the vertical space setting. However, you can set spacing in terms of centimeters, inches, points, picas, and other units of measure. This is useful if you must reserve vertical space of a guaranteed size for inserting artwork, for instance.

To set the vertical space you need, follow **.sp** with a numeric value and the appropriate character indicating the unit of measurement. Some sample settings:

Example	Description
.sp 2i	Sets two inches of vertical space.
.sp 2p	Sets two points of vertical space.
.sp 2v	Sets two lines (two times whatever the **.v** register is set to).
.sp 2	Sets two lines.
.sp 1.5i	Sets a space of 1.5 inches.

The default measurement unit is vertical spaces, **v**. **troff** understands common decimal fractions in most places you might specify numeric values. You can use the same scale factors after **.vs** to define line spacing and with most requests that deal with physical dimensions.

Note: All size numbers are converted to ***machine units***, which are 1/432 inch (1/6 point) horizontally and 1/144 inch (a half point) vertically in **troff**. For most purposes, this is enough resolution that you do not have to worry about the accuracy of the representation. If you need maximum precision of placement, you can specify machine units, with the **u** character following the value. Just remember that machine units are three times larger vertically than horizontally. It is easier to conceptualize in a familiar unit of measure and allow the formatter to make the conversion.

In **nroff**, the machine unit is set dependent on the output resolution of the printer. Generally, **nroff** uses vertical spacing 2 points greater than type size and ignores **.vs** requests. **nroff** uses lines and treats a line as 2 vertical spaces.

2.8.6.2 Reserving Block Space

You can reserve a block of vertical space that cannot be divided by a page break using the **.sv** (save) request. This request otherwise works like the **.sp** request, but you can only give a positive number as a value, and **.sv** does not cause a break.

You can set a trap to reserve a needed amount of vertical space for some purpose, using the **.ne** (need) request:

```
.ne val
```

By default the **val** is 1v. If the vertical space needed cannot be accommodated on the current page, a page break is performed.

2.8.7 Margin Character and Line Numbering

The **.mc** (margin character) request causes the formatter to place a specified character at a specified distance from the right margin of each (nonempty) text line, other than a title line generated by the **.tl** request:

.mc [**char**] [**val**]

The initial **val** is **0.2i** (2/10 inch) in **nroff** and **1m** (1 em) in **troff**. When **val** is not given, the previous **val** is used. If an output line is too long, as can occur when line fill is off, the **char** is appended to the line. The margin character is turned off by using the margin character request with a null **char** (**""**) or by specifying no parameters to the request. This formatter feature is useful for such things as making revision bars.

You can select automatic sequence numbering of output lines with the **.nm** (number-line) request. The form of a line numbering request is:

.nm [**linval** [**multnum** [**znum** [**inum**]]]]

where:

linval is an absolute line value if unsigned or a value relative to the current line value if signed.
multnum is a multiplier. Each line that is a multiple of the value of **multnum** receives a printed line number. Other line numbers are suppressed.
znum is the number of ***digit-spaces*** (spaces the width of a **0** (zero) character in the current font) to separate the text from the line number.
inum is the number of digit-spaces to indent from the current offset for the line number.

The default values are:

Line numbering mode is off
Use current line number plus one for next output line. (When **.nm** is first set, the initial line number is zero.)
Number each line. (Each line is a multiple of one.
Separate text from the line number by a digit-space
Indent the line number from the current offset by zero digit-spaces

If you call **.nm** with no parameters, line numbering is turned off; the default occurs for each parameter not given.

Line numbers are three digits wide, padded on the left by digit-spaces for line numbers that are smaller than three digits. The text lines are thus offset by the default width of four digit-spaces (three digits plus the default one digit-space separation), but they retain their line length. You can reduce the line length to keep the right margin aligned with a preceding right margin. For example, to reduce line length by four digit-spaces, use the request:

\w'0000'u

You can temporarily suspend line numbering with the **.nn** (no number) request:

```
.nn [num]
```

The **num** is the number of lines for which you suppress line numbering. If not specified, the default is **1**. You can also follow an **.nm** with a later **.nm +0** to prevent the current line number from being incremented. If you enter a non-numeric character for a parameter of **.nm**, it is treated as a null-value parameter. For instance:

```
.nm  +5 5 x 3
```

turns on line numbering, with the line number of the next output line to be 5 greater than the last numbered line, with line numbers that are multiples of 5 printed, with spacing between numbers and text unchanged, and with the number indent from offset set to 3.

Blank lines, other vertical spaces, and lines generated by **.tl** requests are not numbered.

2.8.8 Strings and Macros

A ***macro*** is a named set of arbitrary lines that you can invoke by name or with a trap. A ***string*** is a named string of characters, not including a new-line character, that you can call by name at any point. A macro can contain any mixture of text and requests to the formatter. In its simplest form, a macro contains only a string. You can call strings simply by giving a defined string name, but strings can also be handled by normal macros (with or without requests).

Requests, macros, and string names share the same name list. Macros and string names can be one or two characters long and can replace previously defined requests, macros, or strings. You can rename any of these entities with **.rn** (rename) request or remove it with **.rm** (remove) request. You can create macros with the **.de** (define) macro and **.di** (diversion) macro, and append to them using the **.am** (amend) request and the **.da** (divert append) request; **.di** and **.da** cause normal output to be stored in a macro. Strings are named by the **.ds** (define string) request, and a string register holds the string. You can append to a string using the **.as** (append string) request. A macro is called in the same way as a request; a control line beginning with a period and a two-digit macro form a macro call. The remainder of the line can contain up to nine parameters. The contents of string registers of the form **x** and **xx** are inserted in text at any desired point with ***x** and ***(xx** escape sequences respectively. You can nest string names, escape requests, and macro calls within macros.

Subtopics

2.8.8.1 Defining Strings

You can define strings using the **.ds** (define string) request. Defined strings give you a convenient method of inserting strings in text or handling variable strings within macros.

Suppose you are writing a paper containing a large number of occurrences of a degree mark over a letter **A**. Entering:

```
\o'A\\(de'
```

for each occurrence would be tedious. You can instead store an arbitrary collection of text in a string register and thereafter use the string name as a symbol for its contents. The formatter converts the name into the defined string for output. The request line:

```
.ds | \o'A\\(de'
```

defines the string **\o'A\\(de'** to a string register named **|**. Each time **|** appears subsequently in text, the character **Å** is printed at output.

To get Ångstrom (given the above value in the string register **|**), you can enter:

```
\*|ngstrom
```

If a string must begin with blanks, define a string as:

```
.ds xx "   character string with leading blanks
```

where **xx** is a two-character string name that represents the character string following the double quote. The double quote signals the beginning of the definition, and there is no trailing quote; the end of the line terminates the string.

A defined string can actually be several lines long; if the formatter encounters a backslash (\) at the end of any line, it treats the following line as a continuation of the current one. For example:

```
.ds pr "Photographic reproduction by permission of the \
artist.
```

stores the string **Photographic reproduction by permission of the artist.** to a string register, **pr**, which you can insert in text with the escape request:

```
\*(pr
```

You can define strings in terms of other strings or even in terms of themselves.

2.8.8.2 Defining Simple Macros

Suppose you want each paragraph to start in exactly the same way, with a space and a two em temporary indent. You will need a request sequence at the start of each paragraph like:

```
.sp          \"skip (1 line)
.ti +2m      \"indent temporarily (1 line) 2 ems from offset
```

\" is a formatter request that causes the rest of the line to be ignored. It is used to add remarks to input.

To save typing, you can use one short request like **.PP** that will be interpreted by the formatter exactly as:

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

.PP is a macro. You can use the multiline **.de** (define macro) request to define **.PP**:

```
.de PP        \"define a macro PP
.sp
.ti +2m
..            \"end definition of macro
```

.PP is a good name for a paragraph macro, and using uppercase makes it less likely that you will use a name that is already a defined formatter request. (See "User-Definable Names" in topic 2.7.11.3 for a guide to using macro names.)

The definition of **.PP** has to precede its first use. The formatter ignores undefined macros. Names are restricted to one or two characters preceded by a dot (period) or sometimes by an apostrophe (see "Requests That Cause Breaks" in topic 2.8.15.4).

Not only do macros save typing, but they make later changes much easier. Suppose you decide that the paragraph indent is too small, the vertical space is much too big, and Roman font should be forced. Instead of changing the whole document, you only need to redefine **.PP** to something like:

```
.de PP        \"define paragraph macro
.sp 2p
.ti +3m
.ft R
..            \"end (re)definition of macro
```

As another example of macros, consider these two, which start and end a block of offset, unfilled text:

```
.de BS \" start indented block
.sp
.nf
.in +0.3i
..
.de BE \" end indented block
.sp
.fi
.in -0.3i
```

```
..
```

Now you can surround text like:

```
     Copy to
     Lady Marie Antoinette
     Louis XIV, Roi
     Scarlet Pimpernel, Esq.
```

by the macros **.BS** and **.BE**, and it will come out as it did in the preceding example. Notice that the preceding example is indented by **.in +0.3i** instead of **.in 0.3i**, which allows you to nest uses of **.BS** and **.BE** to get blocks within blocks. If you later decide that the indent should be 0.5i (half inch), then simply change the definitions of **.BS** and **.BE**, not the whole text.

Note: The **BS** and **BE** macros are already defined in the memorandum macros, and others in this chapter are also. If you define macros using **.de** (or strings using **.ds**), your definition replaces any previous definition by that label, such as those defined by a called macro package.

To undefine a macro, simply give the **.de** request followed by the name of the macro on the first request line followed with the closing **..** line. You can also remove it using the **.rm** (remove) macro.

2.8.8.3 Defining Macros with Parameters

You can define and repeatedly use macros by following two rules:

1. When you define the macro, construct it so certain parts of it are provided as parameters when the macro is called.
2. Assure that the needed parameters are provided when the macro is called.

This example defines a macro of the name **.SM** that causes **troff** to print something 2 points smaller than the surrounding text:

```
.de SM          \"define SM
\s-2\\$1\s+2    \"shrink the \\$1 string
..              \"end macro definition
```

Within a macro definition, the escape sequence **\\$num** refers to the **num**th string that the macro was called with. Thus, **\\$1** is the string to be placed in a smaller point size when **.SM** is called.

This definition of **.SM** permits optional second and third string parameters that are printed in the normal size. For example:

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

Strings that are not provided when the macro is called are treated as empty strings:

```
.SM BLACK BASS
```

produces:

BLACK**BASS**

```
.SM  FISHING ) (
```

produces:

(FISHING**)**

Trailing punctuation is much more common than leading, so macro parameters are designed to place the little-used third parameter first on output.

The **.$** number register holds the number of parameters a macro is called with.

The following macro **.BD** is the one used to select the bold Roman text that is used for utility names and **troff** request names in text. It combines horizontal motions, width computations, and parameter rearrangement:

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

The **\h** and **\w** requests do not need an extra backslash, as discussed previously. The **\&** is there in case the parameter begins with a period.

Two backslashes are used with the **\\$n** requests, to protect one of them

when the macro is being defined. Perhaps a second example will make this clearer. Consider a macro called **.SH** that produces section headings with the sections numbered automatically and the title in bold in a smaller size. The form of use is:

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

(If the parameter to a macro is to contain blanks, then it must be enclosed by double quotes. This is unlike a string, where only one leading quote is permitted.)

This is the definition of the **.SH** macro:

```
.nr SH 0            \" initialize section number 0 in number register SH
.de SH              \"define SH macro
.sp 0.3i            \"move down 3/10 inch
.ft B               \"set font to bold
.nr SH \\n(SH+1)    \" increment number
.ps \\n(PS-1)       \" decrease PS
\\n(SH.  \\$1       \" number. title
.ps \\n(PS          \" restore PS
.sp 0.3i            \"move down 3/10 inch
.ft R               \"set font to Roman
..
```

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

The preceding definition uses **\n(SH** instead of **\n(PS**. Otherwise, the formatter will get the value of the register at the time the macro is defined, not at the time it is used. Similarly, by using **\\n(PS**, the point size is taken at the time the macro is called. For an example that does not involve numbers, see topic 2.8.14.3.

There is a request that defines macros that discard input; it is the **.ig** (ignore) request:

```
.ig [char1char2]
```

It works exactly like the define request, **.de**, but the macro input is read in copy mode (see "Copy Mode Input Interpretation" in topic 2.8.8.5) and then discarded. Any auto-incremented registers that are used are affected. The two **char** characters, if given, are the characters that end the definition of the macro and are by default period characters (..).

2.8.8.4 Using Macros with Parameters

You can invoke a macro with up to nine parameters on the same line. If the desired parameters do not fit on a line, you can use a concealed new-line (backslash-newline) to continue input on the next line. The parameter separator is the space character, and you can enclose parameters in double quotes to imbed space characters in parameters. A pair of double-quotes imbedded in double-quoted parameters is interpreted as one double-quote character.

When you invoke a macro, the input level is "pushed down" and any parameters available at the previous level are not read until the new level is completely read and the previous level is restored. You can read the macro parameters at any point within the macro using **\$num**, which interprets the **num**th parameter (1**=N=**9) into the current level. If the called parameter does not exist, a null string results. For example, the macro **xx** can 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.

Note that the **\$** was guarded in the definition with a prefixed ****. The number of currently available parameters is held in the **.$** register.

Because string calls are implemented as an input-level push down, no parameters are available from within a string. No parameters are available at the top (non-macro) level or within a trap-called macro.

In ***copy mode***, the formatter copies parameters onto a stack where they are available for reference. This mechanism does not let a parameter contain a direct reference to a long string (converted when the parameter is copied to the stack); you should conceal escape requests with an extra**** to delay conversion until the parameter is referenced.

2.8.8.5 Copy Mode Input Interpretation

During the definition and extension of escape requests and macros (not by diversion), the formatter reads the input in copy mode. The input is copied without interpretation except that:

The contents of number registers indicated by **\n** are extracted.
Strings indicated by ***** are replaced by strings that are the contents of string registers.
Parameters indicated by **\$** are interpreted.
Concealed new-lines indicated by **\(new-line)** are eliminated.
Comments indicated by **\"** are not read.
\t is interpreted as ASCII horizontal tab.
\a is interpreted as an SOH leader character (see "Tabs and Leaders" in topic 2.8.16.2.2).
**** and **\e** are interpreted as ****.
\. is interpreted as "**.**" (a period.).

Padding the above symbols with an additional backslash prefix causes them to be passed through to the next level uninterpreted. (A backslash is stripped away at each pass through a formatter or preformatter.) This works because **** is converted to ****. For example: **\\n** is read as **\n** and is interpreted as a number register indicator when the macro or escape request is reread.

2.8.9 Number Registers and Arithmetic

Number registers, like strings and macros, are useful in setting up a document so it is easy to change later. They also serve for any sort of arithmetic computation. As strings have escape requests, number registers have one- or two-character names you can reference in a line using a **\n** or **\n(** prefix. They are set by the **.nr** (number register) request.

The formatter keeps a set of predefined number registers, among them are **%** 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, respectively; and **.s** and **.f** for the current type size and font.

You can use a number register in computations like any other register, but some numeric registers such as **.s** and **.f**, cannot be changed directly with the **.nr** request.

You can define parameters in terms of the values of a handful of number registers. Some parameters you can control include the point size for text, the vertical spacing of lines, and line and title lengths. To set the point size and vertical spacing for the following paragraphs, for example:

```
.nr PS 9       \"point size = 9 (points)
.nr VS 11      \"vertical spacing = 11 (points)
```

you can define a paragraph macro **.PP**:

```
.de PP         \"set a macro named PP
.ps \\n(PS     \"reset point size
.vs \\n(VSp    \"reset vertical spacing
.ft R          \"set font to 1 (Roman)
.sp 0.5v       \"move down half a vertical space
.ti +3m        \"indent next line 3 ems
..             \"end of macro definition
```

This sets the font to whatever is mounted on the typesetter as font 1 and the point size and line spacing to whatever values are stored in the number registers **PS** and **VS** respectively.

There are two backslashes because if only one backslash is used, point size and vertical spacing are fixed at the time the macro is defined, not when it is used.

You need to buffer only a few requests with an extra backslash:

\n See topic 2.8.9
***** See topic 2.8.8.1
\$ See topic 2.8.8.3
**** The backslash character itself.

Requests such **\s**, **\f**, **\h**, **\v,** and so on do not need an extra backslash, since they are converted by the typesetter to an internal code immediately upon being read.

Subtopics
2.8.9.1 Number Registers
2.8.9.2 Arithmetic Expressions

2.8.9.1 Number Registers

A variety of predefined number registers are available to you as parameters. In addition, you can define your own number registers. Number register names are one or two characters long and do not conflict with request, macro, or string names. Except for certain predefined read-only registers, a number register can be read, written, automatically incremented or decremented, and converted and inserted into the input in a variety of formats. You can use a number register any time numerical input is expected or desired and in numerical expressions (see "Numerical Expressions" in topic 2.8.2.5).

Number registers are created and modified using the **.nr** (number register request), which specifies the name, numerical value, and the auto-increment size. Registers are also modified if accessed with an auto-incrementing escape sequence, **\n**.

You can undefine a number register with the **.rr** (remove register) request. The remove register request has the form:

```
.rr numreg
```

where **numreg** is the one- or two-character number register name. This request can cause problems if the register is needed later.

If the registers **x** and **xx** both contain **num** and have the auto-increment size **incnum**, the following access sequences have the effect shown in the following table:

Table 8-2. Access Sequences

Escape Sequence	Effect on Register	Interpreted Value
\n**x**	None	**num**
\n(**xx**	None	**num**
\n+**x**	**x** incremented by **incnum**	**num** + **incnum**
\n-**x**	**x** incremented by **incnum**	**num** - **incnum**
\n+(**xx**	**x** incremented by **incnum**	**num** + **incnum**
\n-(**xx**	**xx** incremented by **incnum**	**num** - **incnum**

When read, the number register is converted to a value that is decimal (the default value), decimal with leading zeros, lowercase Roman, uppercase Roman, lowercase sequential alphabetic, or uppercase sequential alphabetic according to the format specified by the **.af** request (see **.af** in topic 3.13.4).

2.8.9.2 Arithmetic Expressions

You can use arithmetic expressions anywhere a number is expected:

```
.nr PS \\n(PS-2 \"subtract 2 from value of PS
```

Arithmetic expressions can use the arithmetic operators **+**, **-**,*****, **/**, **%** (mod), the relational operators **>**, **>=**, **<**, **<=**, **=**, and **!=** (not equal), and also parentheses.

The following information provides you with some guidelines for creating arithmetic expressions:

Number registers hold only integers. Formatter arithmetic use truncating integer division (like FORTRAN) and does not round off.

In the absence of parentheses, the formatter evaluates left-to-righ without any operator precedence (including relational operators). Thus, **7*2-4+3/13** evaluates to **1**.

Although integer division causes truncation, each number and its scal indicator is converted to machine units (1/432 inch horizontal, 1/144 inch vertical for **troff**) before any arithmetic is done (so **1i/2u** evaluates to **0.5i** correctly).

Number registers can occur anywhere in an expression, and scal indicators like **p**, **i**, **m**, and so on can also (but spaces cannot).

You must often use the scale indicator **u** (units, not machine units) when you would not expect it, particularly when the formatter is doing arithmetic in a context that refers to horizontal or vertical dimensions. For example, **.ll 7/2i** would seem to mean **3.5 inches**, but actually means **7 ems/2 inches,**; when translated into units, it is truncated to zero. **.ll 7i/2** does not translate to 3.5 inches either; it means 7 inches ÷ 2 ems. The construction **.ll 7i/2u** does evaluate to 3.5 inches. *A safe rule is to attach a scale indicator to every number, even constants.*

If you do arithmetic within a **.nr** request, there is no implication of horizontal or vertical dimension and the default units are "units". In this context, the requests **7i/2** and **7i/2u** mean the same thing. A sample request comparison:

```
.nr ll 7i/2. \"seven inches divided by two (units)
.ll \\n(llu \"line length is value of .n register in units
```

Notice the **u** units indicator on the **.ll** request.

2.8.10 Conditional Tests

The **.if** (if) request creates a built-in test for a condition. It recognizes the following as built-in conditions:

Current page number is odd
Current page number is even
Formatter is **troff**.
Formatter is **nroff**.

Table 8-3. The forms of an .if request

Request Form	Interpretation
.if **c inputstring**	If condition **c** is true, (3) accept **inputstring** as input. (4)
.if !**c inputstring**	If condition **c** is false, (5) accept **inputstring**
.if **num inputstring**	If expression **num** is greater than 0, accept **inputstring**.
.if !**num inputstring**	If expression **num** is less than or equal to 0, accept **inputstring**.
.if **'string1'string2¦inputstring**	If **string1** is identical to **string2**, accept **inputstring**.
.if !**'string1'string2¦inputstring**	If **string1** is not identical to **string2**, accept **inputstring**.

If the condition **c** is true or if the number **num** is greater than zero or if the strings compare identically (including motions and character size and font), **inputstring** is accepted as input.

You can also test for numerical expression and string comparison conditions and for a negation of conditions. You can substitute an **.ie** (if-else) request for an if request when you also offer a corresponding ***else*** request for the reverse conditions of the if-else request. The if-else and else requests must be used as pairs; they can be nested.

An else request is of the form: **.el inputstring**

The formatter remembers the preceding if-else request condition and applies the reverse acceptance condition for the else request. Here are some examples of conditional input acceptance:

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

The preceding prints a title if the page number is even. The following treats page 1 differently from other pages.

```
.ie \n%>1\{\
'sp 0.5i
.tl 'Page % '''
```

```
  'sp|1.2i\}
  .el .sp|2.5i
```

If you use an **if**, **if-else**, or **else** request without a parameter, it is ignored.

(3) **c** is a character for a defined condition: **o** for an odd page condition, **e** for an even page condition, **t** for **troff** formatter condition, and **n** for **nroff** formatter condition.

(4) Spaces between the condition and the beginning of **inputstring** are skipped over. The **inputstring** can be either a single input line (text, macro, or whatever) or a number of input lines. If **inputstring** is larger than one line, the first line must begin with a left delimiter **\{** and the last line must end with a right delimiter **\}**.

(5) The **!** symbol means a negation condition. If a **!** precedes the condition or a numeric or string comparison, the sense of the acceptance is reversed; read **!** as "not".

Subtopics
2.8.10.1 Simple Conditionals
2.8.10.2 Extended Conditionals
2.8.10.3 Comparison Conditionals

2.8.10.1 Simple Conditionals

Suppose you wish the **.SH** macro to leave two extra inches of space just before Section 1, but nowhere else. One way to do that is to test within the **.SH** macro whether the section number is 1 and add space if it is. You can use the **.if** request to provide the conditional test just before the heading line is output:

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

The condition after the **.if** 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 input, either text or a request. If the condition is false, zero, or a negative number, the rest of the line is skipped.

You can make more than one request for a condition by defining a macro **.S1** and invoking it when the condition is true:

```
.de S1
     ...
     ...          \"processing for Section 1 ...
     ...
..
.de SH
.if \\n(SH=1 .S1
     ...
..
```

2.8.10.2 Extended Conditionals

You use an extended form of **.if** to make multiple requests:

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

You must use braces **\{** and **\}** in the positions shown or you will get unexpected extra lines in your output. The formatter also lets you extend if-else constructions.

You can create a negation condition by preceding it with a **!**; you get the same effect as above (but less clearly) by:

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

You can test some conditions with **.if**:

```
.if e .tl "even page title" \"is this page even...
.if o .tl "odd page title" \"or odd?
```

You can give facing pages different titles by using this test inside an appropriate new page macro. A useful test for some purposes determines whether the formatter is **troff** or **nroff**; you can use a different environment or typeface conditionally, for instance.

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

2.8.10.3 Comparison Conditionals

You can make string comparisons in an **.if** statement:

```
  .if 'string1'string2' ...stuff   \"truth test for equality
```

This performs **stuff** if **string1** is the same as **string2**. You can use any character that is not contained in either string or escape request for string separation. The escape requests can reference other strings with *, parameters with \$, and so on.

2.8.11 Diversions

There are numerous occasions in page layout when text is stored for a period of time without printing. Footnotes are an obvious example: the text of the footnote often appears in the input well before its actual output position. In fact, the place where it is output normally depends on its size. There must be a process to examine the footnote and determine its size before printing it.

The formatter provides a mechanism called a ***diversion*** to do this processing. Any part of output can be diverted into a macro instead of being printed, and then inserted into the input at a convenient time.

The **.di** (diversion) request begins a diversion:

```
.di xy
```

All subsequent output is collected into the macro named **xy** until the request **.di** is encountered with no parameters, thus, ending the diversion. The processed text is available at any time thereafter, simply by summoning the request diversion macro:

```
.xy   \"read the defined diversion macro xy
```

The vertical size value of the last finished diversion is stored in the built-in **.dn** diversion number register.

As a simple example, suppose we want to implement a ***keep-release*** operation, so that material between the requests **.KS** and **.KE** is not split across a page boundary (as for a figure or table). When the formatter reads the **.KS** (keep start) request, it begins diverting output until it sees the **.KE** (keep end) request. It determines whether the diverted text will fit on the remaining space on the page, or whether the text must go on the next page to keep it intact:

```
.de KS                      \" define the start keep
.br                         \" start fresh line
.ev 1                       \" collect in environment 1
.fi                         \" make it filled text
.di XX                      \" divert it and store in XX
..                          \"end of definition of KS
.de KE                      \" define the end keep
.br                         \" get last partial line
.di                         \" end the diversion
.if \\n(dn>=\\n(.t .bp      \"page break if XX doesn't fit
.nf                         \" return to no-fill of text
.XX                         \" insert the diverted text
.ev                         \" return to normal environment
..
```

Recall that the **.nl** number register is the current position on the output page. Since output was being diverted, this remains at its value when the diversion started; **dn** represents the amount of text in the diversion; **.t** (another built-in register) is the distance to the next trap, which 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** (begin page) request is issued. In either case, the diverted output is brought back with the macro **.XX**. It is essential to bring it back in no-fill mode so the formatter will do no further processing on it.

You can divert processed output 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. You can set a single diversion trap 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 vertical space value. Constant-spaced or bold text that is diverted can be reread correctly only if these modes are still (or again) in effect at reread time. One way to assure this is to imbed the appropriate **.cs** or **.bd** requests in the diversion using the function described under "Transparent Throughput" in topic 2.8.16.3.3.

Subtopics
2.8.11.1 Nesting
2.8.11.2 Traps

2.8.11.1 Nesting

You can nest diversions (the top nondiversion level is called the ***0th diversion level***), and certain parameters and registers are associated with the current diversion level. These are:

The diversion trap and associated macr
No-space mod
The internally saved marked place (see **mk** and **rt**)
The current vertical place (held in **.d** register)
The current high-water text base-line (held in **.h** register)
The current diversion name (held in **.z** register).

2.8.11.2 Traps

A **.wh** (when) request tells the formatter to read a macro when a certain condition is met. The formatter provides three types of trap mechanisms:

Page trap
Diversion trap
Input-line-count traps

You can place macro-calling traps at any page position including the top, using the **.wh** request, and change the position using the **.ch** request. Trap positions at or below the bottom of the page have no effect unless they are moved to within the page or rendered effective by increasing the page length. You can plant two traps at the same position only by first planting them at different positions and then moving one of the traps. The first trap planted will conceal the second, unless the first one is moved. If the first one is returned to its original position, it again conceals the second trap. The formatter invokes the macro associated with a page trap automatically when the vertical size of a line of text reaches or extends past the trap position. Reaching the bottom of a page springs the top-of-page trap, if there is a next page. The distance to the next trap position is held in the **.t** register. If there are no traps between the current position and the bottom of the page, the value in **.t** is the distance to the page bottom. For an example of a trap, see "New Page Trap" in topic 2.8.14.1.1.

You can place a macro-calling trap in the current diversion using the **.dt** request. The **.t** register works in a diversion and gives a "large" distance value if there is no subsequent trap. Another **dt** request redefines the diversion trap; if you give no parameter for **.dt**, the trap is removed.

2.8.12 Environment Switching

You can control a number of text processing parameters together in an ***environment*** that you can switch where appropriate. Environment parameters are initialized with default values, and you can switch environments with the **.ev** (environment) request:

```
.ev [num]
```

The formatter does environment switching by pushdown; a previous environment is stored and can only be returned by using an **.ev** request. The initial value of **num** is 0 and cannot be greater than 2. You can have up to three environments, by invoking environment 1 from environment **0**, and environment 2 from environment 1. To recall environment **0** from environment 2 (or visa versa), you must first pass through environment 1. Collected partial lines and words are passed to a new environment and each of the requests in the following list can be passed to a new environment. Everything else is global, such as page-oriented parameters, diversion-oriented parameters, number registers, and macro and string definitions.

Table 8-4. Environmentally Controlled Requests

.ad	**.cc**	**.ce**	**.cu**	**.c2**	**.fi**	**.ft**	
.hc	**.hy**	**.in**	**.it**	**.lc**	**.ll**	**.ls**	
.lt	**.mc**	**.na**	**.nh**	**.nf**	**.nm**	**.nn**	
.ps	**.ss**	**.ta**	**.tc**	**.ti**	**.ul**	**.vs**	

A formatting ***environment*** provides a package of settable parameters for doing text formatting, such as type size, font, line and title lengths, line-fill on or off, tab stop settings, and partial-line handling parameters. The formatter can store three environments you can call to conveniently handle certain kinds of problems without keying in a large number of conditional statements. As was mentioned earlier, there is a potential problem when crossing a page boundary: parameters like size and font for a page title may well be different from those in effect in the text when the page boundary occurs. This titling problem is solved by processing the main text in one environment and titles in a separate environment, with each environment containing its own parameters.

The request **.ev num** shifts to environment **num**; **num** must be **0**, **1**, or **2**. The initial environment is **0** when you summon the formatter. The **.ev** request given with no parameter returns the previous environment. Environment names are maintained in a stack. Therefore, calls for different environments can be nested and called in explicit sequence.

Most commonly, main text is processed in environment **0**, where the formatter begins. You can modify the **.NP** (new page) macro to process titles in environment **1** like this:

```
.de NP          \"define a new page macro
.ev 1           \" shift to environment one
```

```
.lt 6i              \"now setting environment parameters
.ft R
.ps 10
   ... any other processing ...
.ev                 \" return to previous environment, saving this one
..
```

You can also initialize environment parameters outside the **.NP** macro, but the method shown keeps all the processing in one place and is thus easier to understand and change (assuming you consistently use environment 1 for a page header environment).

2.8.13 Page Control

Page top and bottom margins are not automatically provided; normally you define two macros and set page traps for them at vertical positions **0** (top-of-page) and some **-num** from the bottom. (This area is more easily controlled using the memorandum macros rather than directly using a formatter. See Chapter 7, "Memorandum Macros.") A pseudo-page transition onto the first page occurs either when the first break occurs or when the first nondiverted text processing occurs. You must prepare the top-of-page trap before this transition. The following references to the ***current diversion*** mean that the mechanism being described works during both ordinary and diverted output (the former considered as the top diversion level). (See "Diversions" in topic 2.8.11.)

The usable page width on the C/A/T phototypesetter is about 7.54 inches, beginning about 1/27 inch from the left edge of the 8-inch wide continuous-roll machine paper. The physical limitations on **nroff** output are output-device-dependent.

The text length of the last line output is stored in the **.n** register, and text base-line position on the page for this line is stored in the **nl** register. The lowest place text appears on the current page is stored as a value in the **.h** register.

The **.bp** (begin page) request breaks the current line, ejects the current page and begins a new page, and is of the form:

```
.bp [±num]
```

where **num**, if given, is the new page number.

You can set output page length using the **.pl** (page length) request:

```
.pl [±val]
```

where **val** is a scaled value (11i default). The current page length is stored in the **.p** (page length) number register. The **nroff** formatter uses page length in terms of lines of text.

See "Page Headers and Footers" in topic 2.8.14 for further information on page control features.

Subtopics
2.8.13.1 Line Spacing
2.8.13.2 Line Separation
2.8.13.3 Extra Line Separation
2.8.13.4 No Space Mode
2.8.13.5 Line Lengths
2.8.13.6 Margins and Indents

2.8.13.1 Line Spacing

You can set the number of lines to space using the **.ls** (line spacing request):

```
.ls num
```

The **num** parameter is an unsigned whole number greater than zero. The number of lines skipped is equal to **num**-1. Initial line spacing is set to **1**, resulting in single-spaced output (no lines skipped). You can select double spacing with the **..ls 2** request. The number parameter indicates the number of whole lines to allow for each line printed.

2.8.13.2 Line Separation

Another factor that determines what the typeset text looks like is the spacing between lines, which is set independently of the point size. Vertical spacing is measured from the bottom of one line to the bottom of the next. The request to control vertical spacing is **.vs**. For running text, you will generally get pleasing results when you set the vertical spacing roughly 20 percent larger than the character size. For the popular type sizes, the rule-of-thumb is a vertical spacing two points larger than the selected typeface. By default, **troff** uses ***10 on 12 type***; that is 10-point type with 12-point vertical spacing:

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

The **.vs** request without a following value causes **troff** to revert to the previous vertical spacing. The vertical spacing value is initialized by the formatter and held in the **.v** register. The default value is type size plus two points.

2.8.13.3 Extra Line Separation

If a word contains a vertically tall construct requiring the output line containing it to have extra line separation before or after it, you can use the **\x** (extra space) request of the form:

\x'**num**'

The escape request is imbedded in or attached to the word. In this and other functions having a pair of delimiters around their parameter (here '), you can select an arbitrary delimiter, except that it cannot look like part of the number **num** (periods, for instance, cannot be used). If **num** is negative, the output line containing the word is preceded by **num** extra vertical space; if **num** is positive, the output line containing the word is followed by **num** extra vertical space. If successive requests for extra space in a line apply to the same line separation, the largest value is used. The most recently used post-line extra line-space is stored in the **.a** register.

2.8.13.4 No Space Mode

The **.ns** (no space) request turns on the no space mode. This mode inhibits **.sp** (space) requests and **.bp** (begin page) requests until a line of output occurs or the **.rs** (restore spacing) request is called, returning the spacing to the previous spacing mode.

2.8.13.5 Line Lengths

The **troff** formatter provides a default line length of 6.5 inches. The **nroff** formatter assumes a 65-space line length in 10-point print or a 72-space line length in 12-point print. As with the **.sp** request, you can specify length in several scales. To set the line length, use the **.ll** request, as in **.ll 6i**, which sets line length to 6 inches.

Note: The maximum line length **troff** can send to the typesetter is about 7.54 inches.

You can set the indent level with the **.in** request. The **.ce** request (see topic 2.8.15.4) centers text using the line length minus the indent level.

The current line length is stored in the **.l** number register; the current indent level is stored in the **.i** register. (The length of a three-part title is stored in the register **tl** and is set independently of line length and indent.) The effect of an **.ll**, **.in**, or **.ti** request is delayed until after any partially collected line is output. (See "Titles" in topic 2.8.14.1.)

2.8.13.6 Margins and Indents

The left margin, or ***page offset***, is set using the **.po** (page offset) request. The default **nroff** offset is **0**; the default **troff** value is approximately 26/27 inch, giving 1 inch total offset after inclusion of 1/27 inch typesetter mechanical offset. The request is of the form:

```
.po [±val]
```

where **val** is a scaled value. The current page offset is stored in the **.o** (offset) number register.

You can set the left margin further in from page offset using the **.in** (indent) request. You can create blocks of variously offset text using **.in** to move the left margin in and **.ll** to set the line length, which effectively sets the right margin. For example, the text below is created with the code that follows it:

> In its earliest development knowledge is self-sown. Impressions force themselves upon men's senses whether they will or not, and often against their will. The amount of interest which these impressions awaken is determined by the coarser pains and pleasures which they carry in their train, or by mere curiosity; and reason deals with the materials supplied to it as far as that interest carries it, and no further. Such common knowledge is rather brought than sought; and such ratiocination is little more than the working of a blind intellectual instinct. (6)

```
.in +2i     \"move the left margin 2 inches right
.ll -2i     \"shorten line length to hold right margin
In its earliest development knowledge is self-sown.
Impressions force themselves upon men's senses whether
they will or not, and often against their will.
The amount of interest which these impressions awaken
is determined by the coarser pains and pleasures which
they carry in their train, or by mere curiosity;
and reason deals with the materials supplied to it as far
as that interest carries it, and no further.
Such common knowledge is rather brought than sought;
and such ratiocination is little more than the working of a
blind intellectual instinct.
.ll +2i     \"return the line length to normal
.in -2i     \"and the left margin too
```

Notice the use of **+** and **-** to specify the amount of change relative to the margin. These requests change the previous setting by the specified incremental amount, rather than just overriding it. The distinction is important:

```
.ll +1i
```

makes lines 1 inch **longer** than they were set.

```
.ll 1i
```

makes lines 1 inch **long**.

With **.in**, **.ll**, and **.po**, the formatter returns to the previous value if no

parameter is given.

(6) From *The Crayfish, T.H. Huxley, D. Appleton publishers, 1880*.

Subtopics
2.8.13.6.1 Temporary Margin Changes and Indenting

2.8.13.6.1 Temporary Margin Changes and Indenting

To indent a single line, use the **.ti** (temporary indent) request. To indent paragraphs, you begin all paragraphs with a request like:

```
.ti 3
```

The default unit for **.ti**, as for most horizontal movement requests (**.ll**, **.in**, **.po**), is ***em***; an em is the width of the letter m in the current type and point size; an em is as wide as the type size in points (an em is 10 points wide in 10 point type, for instance). Although fixed units of measure (inches, centimeters) are usually clearer than ems, the em has an important use; it is a scale of line size that is proportional to the current point size. If you want to make text that keeps its proportions regardless of point size, you should use the em for horizontal measures. You can let horizontal measures use the default specification of em, or specify em directly, as in **.ti 2.5m**.

You can indent negatively if the offset is already positive.

2.8.14 Page Headers and Footers

The features described in this section are all optional formatter features. You must make requests for their use if you want to use their features. The formatter **-mm** command line flag calls these features with defaults established by defining and using the memorandum macros. (See Chapter 7, "Memorandum Macros.")

Subtopics

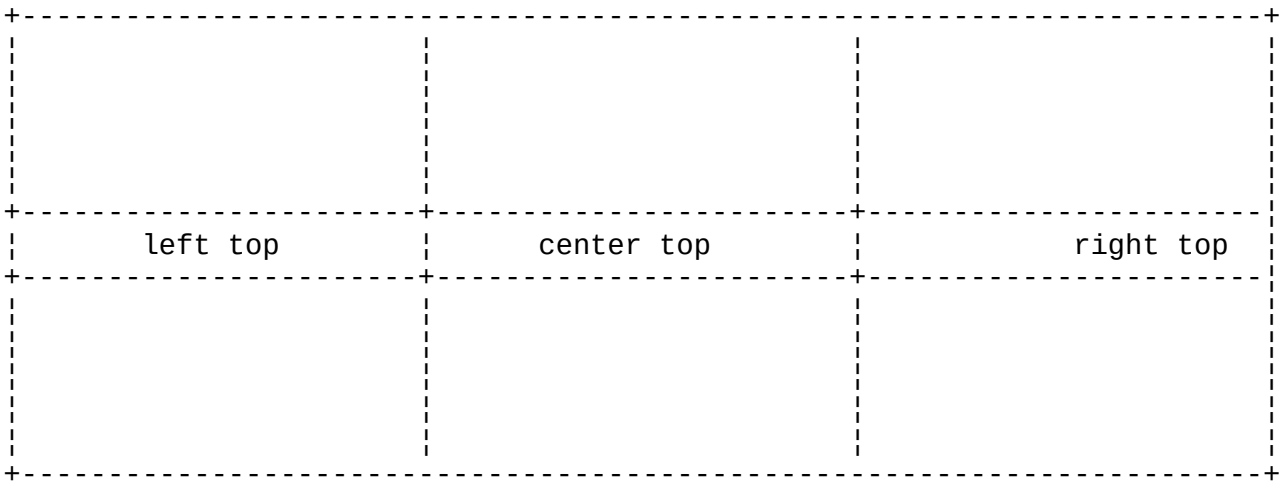
2.8.14.1 Titles

The formatter **.tl** (title) request provides for automatic placement of three title fields at the left, center, and right of a line:

```
.tl'left'[center'[right']]
```

with a title-length specifiable with the **.lt** (title length) request. The strings **left**, **center**, and **right**, are respectively left-adjusted, centered, and right-adjusted fields in the current title-length. Any of the strings can be empty, and fields can overlap. You can use **.tl** anywhere in input; it is independent of the normal text collecting process. The title length value (**.lt**) is independent of line length settings, but the title does use the current page offset.

Suppose you want the following at the top of each page:



You must enter what the actual title is and when to print it, as well as what to do at and around the title line. First define a new page macro to process titles and page headers and footers at the end of one page and the beginning of the next:

```
.de NP
'bp                                    \"begin page request
'sp 0.5i                               \"space down .5 inch
.tl 'left top'center top'right top'    \"title labels
'sp 0.3i                               \"space down .3 inch
..
```

Subtopics
2.8.14.1.1 New Page Trap
2.8.14.1.2 Page Number Character

2.8.14.1.1 New Page Trap

A **.wh** (when) request causes the formatter to read a macro when a certain condition is met (see "Traps" in topic 2.8.11.2). You want the formatter to use the new page macro when, for example, its output text gets 1 inch from the bottom of a page:

```
.wh -1i NP      \"when (page) -1 inch is reached, new page
```

NP is not preceded by a period; this is simply the name of a macro, not a macro call. The minus sign means **measure back from the page bottom**, so **-1i** can be read as **1 inch from the bottom**.

The **.wh** request is used outside the definition of **.NP**:

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

As text is output, the formatter keeps track of its vertical position on the page. After a line is printed within 1 inch from the bottom, the **.NP** macro is activated. **.NP** causes a skip to the top of the next page with no break (that is what the **'bp** was for) then prints the title with the appropriate margins. Page formatting is handled by a series of traps (see "Traps" in topic 2.8.11.2).

2.8.14.1.2 Page Number Character

You can call the current page number in a title field by using the ***page number token*** (by default, **%**). The formatter inserts the contents of the **%** page number register when it finds the page number token in a title field; you can change the character with the **.pc** (page-number character) request to any character:

.pc [**char**]

2.8.14.2 Title Line Length

The length of a title line is independent of the current line length; therefore, titles come out at the default length of 6.5 inches unless you change it with the **.lt** (title length) request.

There are several ways to fix problems of point sizes and fonts in titles. For simple applications, you can change **.NP** to set the proper size and font for the title. 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'center'right'        \"give the title line
.ps                            \" return to previous point size
.ft P                          \" return 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. Changes of that nature are handled through changing environments.

2.8.14.3 Titles with Macro Fields

The preceding **.NP** macro definition has a title line:

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

You can make these fields into macro parameters by using:

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

The title now is assembled using the contents of three string registers named **LT**, **CT** and **RT**. Each field that is empty will be blank. Normally CT is set to something like:

```
.ds  CT -%-          \" Set center title as page number between hyphens
```

(See "Page Numbering" in topic 2.8.14.5.) You can, of course, define any of the strings.

2.8.14.4 Footers

To get a footer at the bottom of a page, you can modify **.NP** so it does some processing before the **'bp** request, or you can split the job into a footer macro called at the bottom margin and a header macro called at the top of the page.

2.8.14.5 Page Numbering

Page numbers are assigned by the formatter automatically as each page is produced (starting at 1), but no numbers are printed unless you ask for them explicitly. To print page numbers, include the key character **%** in the **.tl** line at the position where you want the number to appear. For example:

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

centers and places the page number inside hyphens. You can reset the page number at any time with either **.bp num**, which starts a new page numbered **num**, or with **.pn num**, which sets the page number for the next page but does not cause a skip to the new page. Again, **.bp +num** sets the page number to **num** more than its current value; **.bp** means **.bp +1**. **.pn** stores the value of the current page number in the **%** number register.

2.8.15 Motion Control

There are a number of requests that allow motion control.

Subtopics
2.8.15.1 Local Motions: Drawing Lines and Characters
2.8.15.2 Width Request
2.8.15.3 Special Local Motion Requests
2.8.15.4 Requests That Cause Breaks

2.8.15.1 Local Motions: Drawing Lines and Characters

troff responds to a host of requests for placing characters of any size at any place in text. You can use them to draw special characters or to tune your output for a particular appearance.

Subtopics

2.8.15.1.1 Local Vertical Motions

To avoid unexpected vertical dislocations, you must assure that the net vertical local motion within a word in filled text, and otherwise within a line, balance to zero. The following escape requests give local vertical motion:

Escape Request	Effect of the Escape Request
\v'num'	Moves the distance **num**, where **num** is a scaled number recognized by the formatter.
\u	In **troff**, moves half an em up. In **nroff**, moves half a line up.
\d	In **troff**, moves half an em down. In **nroff**, moves half a line down.
\r	In **troff**, moves one em up. In **nroff**, moves one line up.

The equation E=MC(2) is generated by the request sequence:

```
E=MC\s-2\v'-0.4m'2\v'0.4m'\s+2
```

In this example, the 0.4 em vertical motion specified is in the smaller size specified for the numeral 2.

2.8.15.1.2 Local Horizontal Motion and ASCII Backspace

The following escape requests give horizontal local motion:

Escape Request	Effect of the Escape Request
\h'num'	Moves the distance **num**, where **num** is a scaled number recognized by the formatter.
|	In **troff**, moves 1/6 em space to the right. Ignored by **nroff**.
\(^)	In **troff**, moves 1/12 em space to the right. Ignored by **nroff**.

Unless in copy mode, the formatter replaces the ASCII backspace character with a backward horizontal motion having the width of the space character. The backspacing function is used for overstriking characters, underlining, and similar purposes.

2.8.15.1.3 Mark Horizontal Place

The **\k** (mark) escape request stores the current horizontal position in the input line to be stored in a register **x** that follows:

```
\kx
```

The following construction causes **word** to print in bold type by backing up almost to its beginning and overprinting it with a slight offset:

```
\kxword\h'|\nxu+2u'word
```

This form of emboldened type is called emphasized print. The above construction is read by the formatter as:

Stores current position to register **x** (**\kx**).
Passes **word** to output as text (**word**).
Moves horizontally to the location marked in register **x** plus two basic units (**\h'|\nx+2'**).
Passes **word** to output as text (**word**).

2.8.15.1.4 Specific Horizontal Movement

You can specify an absolute horizontal motion. The **\h** request is like the **\v** request, except that the default scale factor is ems instead of line spaces. For example:

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

causes a backwards motion of 1/10 inch. The **eqn** preprocessor performs some adjustments for you automatically.

Frequently **\h** is used along with the **\w** (width) request 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 (1/432nds of an inch). All formatter computations are ultimately done in these units. To move horizontally the width of an x, you can say:

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

The default scale factor for all horizontal dimensions is the em; therefore, the **u** is specified for machine units. You can nest quotes as much as you like, as long as you keep them balanced in pairs.

2.8.15.2 Width Request

The **\w** (width) request sets the width of a given string. For **troff**, it also sets the height of the given string. The width request:

```
\w'string'
```

generates the numerical width of **string** in basic units. For example:

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

temporarily indents to the left a distance equal to the length of the **1.** string. You can imbed size and font changes in **string** without affecting the current environment.

The width request also sets three number registers that **troff** uses. The formatter sets the **st** (string top) register and the **sb** (string bottom) register to the highest and lowest extent of **string** relative to the baseline, respectively. The total height (in basic units) of the string is the value stored in **st** minus the value (zero or negative) stored in **sb**. The number register **ct** is set to a whole number value from **0** to 3.

Value	Effect of Value
0	All of the characters in **string** are short lowercase characters without descenders (like the character e).
1	At least one character in **string** has a descender (like the character y).
2	At least one character in **string** is tall (like the character H).
3	Both tall characters and characters with descenders are present in **string**.

The **troff** formatter uses the value in the **ct** register to assure an even physical separation of each line of output text.

2.8.15.3 Special Local Motion Requests

There are several special-purpose **troff** requests for local motion. For example, the **\0** request creates a ***constant space*** of the same width as a digit. A constant space is never widened or split across a line by line justification and filling. These are constant spaces:

\<blank> Equals the width of a space

\| Equals half the width of a space

\(^) Equals one quarter the width of a space

\& Has zero width. This is used, for example, to keep a line of text that begins with a period from being treated by the formatter as a request line.

The **\o** (overstrike) request used in the form:

\o'**characters**'

can be used to cause up to 9 **characters** to be overstruck, centered on the widest character. This is particularly useful for accents.

You can create overstrikes with another special convention, the **\z** (zero-motion) request:

\z**x**

This suppresses the normal horizontal motion after printing the single character **x**, so another character can be laid on top of it, aligned at the left edge. Although you can change sizes within an overstrike request, **\o** centers the characters on the widest given character, and there can be no horizontal or vertical motions. **\z** may be the only way to get what you want.

You can create an elaborate overstrike sequence by using the **\b** (bracketing) request, which builds characters vertically and keeps them centered on the current baseline.

Subtopics

2.8.15.3.1 Half-Line Scrolling

If you are not using the **eqn** preprocessor, subscripts and superscripts are most easily done with half-line scrolling requests **\u** and **\d**. To move up on the page half a point size, insert a **\u** at the desired place; to move down, insert a **\d**. You should always use **\u** and **\d** in pairs. Thus:

```
  Area = \(*pr\u\s-42\s0\d
```

produces:

Area = &pi.r(2)

Enclosing the superscript numeral with **\s-4...\s0** makes the numeral 4 points smaller than the current point size. Since the **\u** and **\d** requests both refer to the current point size, be sure to put them both inside or outside the size changes, or you will get an unbalanced vertical motion. (**troff** is particularly sensitive to unbalanced changes that span more than one line.)

2.8.15.3.2 Scaled Vertical Movement

Sometimes the movement given by **\u** and **\d** requests is not the amount you desire. You can use the **\v** request to specify the amount of vertical movement. The **\v'num'** escape request causes motion up or down the page by the amount specified in **num**.

A minus sign directs upward motion, while no sign or a plus sign directs **troff** downward motion. Thus, **\v'-2'** causes an upward vertical motion of two line spaces.

There are many other scales of motion and ways to express them. The requests:

```
\v'0.li'
\v'3p'
\v'-0.5P'
```

and others are understood by **troff**. Notice that the scale specifier in the preceding examples is set inside single quotes. Other characters can be used in place of the quotes; this is true of all other **troff** requests described in this section.

2.8.15.4 Requests That Cause Breaks

Some formatter requests, such as **.bp** and **.sp**, cause a ***break*** to take place. That is, all input text collected but not yet printed is printed as soon as possible, and the next input line starts a new line of output. If you use **.sp.** or **.bp** in the **.NP** macro, this causes a break in the middle of the current output line when a new page is started. The effect for the formatter is to print the remainder of the last line that was read before performing the macro and place the next input line at the start of a new output line. This is not necessarily what you want. Using the ' instead of **.** prefixing a request line tells the formatter that no break is to take place. The output line currently being filled is not be forced out before the macro is performed.

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

.bp num ***break page*** Ejects current page, numbers the next page **num**.

.br ***break*** Breaks immediately.

.ce num ***center*** Centers the next **num** lines of input.

.fi ***fill*** Fills output lines.

.nf ***no fill*** Does not fill or adjust output lines.

.sp num ***space*** Moves the vertical distance in the amount and direction specified by **num**, where a negative number is backward (upward) movement.

.in num ***indent*** Moves the left margin by the amount **num**.

.ti ***temporary indent*** Moves the left margin by the amount **num** for the next line.

The breaking action of the above requests can be suppressed by substituting a ' for the **.** prefix. If you really need a break, add a **.br** request at the appropriate place.

Note: If you change fonts or point sizes frequently, you may find (if you cross a page boundary in an unexpected font or size) that your titles come out in that size and font instead of what you intended.

2.8.16 Character Translations

Characters can be converted for use within the formatter or translated for output.

Subtopics
2.8.16.1 Non-printing ASCII Characters
2.8.16.2 Escape Requests
2.8.16.3 Character Translation for Output
2.8.16.4 Concealed New-Lines and Comment Lines

2.8.16.1 Non-printing ASCII Characters

You can use the ASCII characters STX, ETX, EMQ, ACK, SO, SI, and ESC as delimiters or have them translated into another character or graphic with a **.tr** request (see topic 2.8.16.3).

Note: You can also use or translate the ASCII BEL character, but the memorandum macro package uses the BEL character in some of its macros. If you define macros using the BEL character or translate a character to the BEL character and also use the memorandum macros, macro definition conflicts may cause unexpected results.

The **troff** formatter normally passes none of these characters to its output, but it passes all eight characters if you invoke it with the **-a** command-line flag). **nroff** passes the last three and the BEL character. Their effect depends on the output device used. Characters that are not passed to output are ignored.

2.8.16.2 Escape Requests

The backslash character **** causes the following character to call the contents of a string register or introduces some request in the form of an escape sequence. You can define a different character than the backslash as an escape character with the **ec** (escape character) request. The escape request **\e** passes whatever the current escape character is as literal output. If necessary or convenient, you can turn the escape mechanism off with **.eo**; **.ec** restores it.

Subtopics
2.8.16.2.1 Request Line Prefixes
2.8.16.2.2 Tabs and Leaders
2.8.16.2.3 Field Delimiters

2.8.16.2.1 Request Line Prefixes

You can change the period control character and the accent mark (apostrophe) ***no-break control character*** that are used as prefixes to indicate the beginning of a request line. The change must be compatible with the design of any macros used in the portion of the input where the change is in effect, particularly if any trap-called macros are involved. The request:

```
.cc [char]
```

sets the character **char** as the new control character replacing the . (period) character. If **char** is omitted following a **.cc** request, the period character is restored as the control character. For instance, if **char** was made **>** by a previous **.cc** (control character) request, then the following request restores the period as the control character:

```
>cc
```

The no-break control character is changed with a second control character request, **.c2**, which works in the same way as **.cc**.

2.8.16.2.2 Tabs and Leaders

Tabs (ASCII horizontal tab character HT) are generally used only in ***unfilled text***. This is text that is formatted as it was entered rather than reformatted to make each line contain as many words as possible. By default, tabs generate motion and leaders (ASCII leader character SOH) generate a string of periods; **.tc** (tab character) and **.lc** (leader character) requests offer the choice of repeated character or motion. Tab stops are set by default every half inch from the current indent; you can change them using the **.ta** (tab) request. To set stops every inch, enter:

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

Tabs are left-aligned by default, but they can also be right or center-aligned by using the **R** or **C** key letters. To set a left-adjusted tab stop at 1 inch, a right-adjusted tab at 2 inches, and a centered tab at 3 inches, use the **.ta** request:

```
.ta 1i 2iR 3iC
```

or

```
.ta 1iL 2iR 3iC
```

If you have a more complicated table layout, such as columns of numbers to be aligned at their decimal points, the **tbl** preprocessor provides a variety of functions.

You can fill up tabbed-over space with a character other than a blank by setting the tab character with the **.tc** request:

```
.ta 1.5i 2.5i
.tc \(ru              \" \(ru is "_"
Name tab Age tab
```

To produce:

```
Name ------------ Age ------
```

To reset the tab substitute character to a blank, use **.tc** without a parameter. (You can draw lines with the **\l** request; see "Local Motions: Drawing Lines and Characters" in topic 2.8.15.1.) There are three types of internal tab stops: left adjusting, right adjusting, and centering. In the following table:

dval is the distance from the current position on the input line (where a tab or leader was found) to the next tab stop.
nstring consists of the input characters following the tab (or leader) up to the next tab (or leader) or end of line.
wval is the width of **nstring**.

Tab Type	**Length Of Motion Or Repeated Character**	**Location of Next String**
Left	**dval**	Following **dval**
Right	**dval** - **wval**	Right adjusted within **dval**

Centered	**dval** - **wval**/2	Centered on right end of **dval**

The length of generated motion can be negative, but a repeated character string length cannot be. Repeated character strings contain an integer number of characters, and any residual distance is taken as motion at the start of the string. Tabs or leaders found after the last tab stop are ignored but can be used as **nstring** terminators.

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

2.8.16.2.3 Field Delimiters

A ***field*** is contained between a pair of ***field delimiter*** characters and 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 distributed as horizontal padding space among the indicated padding places. The padding can be negative. For example, if the field delimiter is # and the padding indicator is (^), the following specifies a right-adjusted string with the **string** centered in the remaining space:

```
#(^)string(^)right#
```

2.8.16.3 Character Translation for Output

You can cause one character from input to appear as another at output using the **.tr** (translate) request. All character comparisons take place with the input (stand-in) character, which appears to have the width of the final character. The graphic translation occurs only at the moment of output (including diversion). The translate request is of the form:

```
.tr abcd...
```

The second character (**b**) is output where the first (**a**) is input, the fourth is output (**d**) where the third (**c**) is input, and so on. If **.tr** receives an odd number of parameters, the last character given is replaced with a space character at output.

Subtopics
2.8.16.3.1 Translated Characters
2.8.16.3.2 Ligatures
2.8.16.3.3 Transparent Throughput

2.8.16.3.1 Translated Characters

The formatter (particularly **troff**) automatically translates some characters into others:

` grave accents and ' acute accents become open and close singl quotes respectively (' ').
A pair of double quotes ("...") are changed to left and right doubl quotes ("...").
A - (minus sign) is treated as a hyphen (-)
A **** (backslash) character is interpreted as an escape (inline) request. You can specify explicit characters by preceding the character with the escape character, ****.

Note: To get an explicit backslash, input **\e** rather than \\. You should reserve multiple backslashes for character padding purposes when you wish to protect something through multiple processing, such as processing by **eqn**, **tbl**, and **troff**.

nroff accepts the entire **troff** character set as input but can print only ASCII characters, special characters that the particular printer is mapped to support, and characters that are defined by assembling them from the character set by overstriking. Some printers can permit custom character definition by a programmer.

You can change what conversions are performed using the **.tr** (translate) request. For example, to change a dash to an underscore, use this request:

```
.tr \(mi\(em
```

The formatter reads this as **translate - into** _.

You do not generally select the special font; **troff** uses it automatically as needed. (**troff** gets characters @, #, **"**, `, **'**, **<**, **>**, \, **{**, **}**, **~**, (^), &, and _ from the special font. If the special font is not mounted, **troff** outputs a space 1 em wide each time it receives one of these characters as input.)

2.8.16.3.2 Ligatures

When ligature mode is set, five ligatures are available in the current **troff** character set: fi, fl, ff, ffi, and ffl. **nroff** understands the ligature requests, converting them to the Roman character string each ligature represents, as does **troff** when ligature mode is off.

Ligature mode is called with the **.lg** request:

```
.lg [num]
```

Ligature mode is turned on if **num** is absent or nonzero and turned off if **num** is **0**. If **num** is 2, only the two-character ligatures are automatically called. The ligature mode is normally on in **troff** and automatically invokes ligatures during input. **nroff** ignores the **.lg** request. Ligature mode is inhibited for request, macro, escape request, string or file names, and copy mode.

The requests for the ligatures are:

Table 8-5. Ligatures

Escape Request	Roman Characters	Ligature
\(fi	fi	fi
\(fl	fl	fl
\(ff	ff	ff
\(Fi	ffi	ffi
\(Fl	ffl	ffl

2.8.16.3.3 Transparent Throughput

The formatter reads an input line beginning with a **\!** in copy mode and transparently outputs it without the initial **\!**. The formatter is otherwise unaware of the line. You can pass control information to a postprocessor or imbed control lines in a macro created by a diversion using this special line control request. (See "Diversions" in topic 2.8.11.)

2.8.16.4 Concealed New-Lines and Comment Lines

You can split a long input line that must stay one line into many physical lines by ending all but the last one with the **** escape character. The sequence ****<new-line> is not output except when it appears within a comment. Any text appearing on a line to the right of a **\"** escape request is treated by the formatter as a comment and is not output. The formatter does read the new-line at the end of a comment. A line beginning with **\"** appears as a blank line and functions like a **.sp 1** (space) request.

2.8.17 Passing Commands to the System

You give a system command by preceding it with the **.!** request on an input line:

.! **command**

After the command is performed, the formatter continues to process where it left off.

2.8.18 Input and Output Handling

You can alter the processing that the formatter performs by inserting input control requests in the input file.

Subtopics

2.8.18.1 File Switching and Piping

You can use the **.so** (source) request to switch source files and cause formatting of a file other than the one currently being processed. A source request is of the form:

.so **file**

where **file** is the file to switch to. When **file** is finished or exited, processing of the prior file resumes where it was left off. Effectively, **file** is inserted into the file the formatter is processing at the point when **file** is called; macros, requests, and escape requests are interpreted as they are encountered within **file** rather than when **file** is exited. You can nest source requests. Each **.so** request must have a **file** parameter.

You can switch files to another file without returning to the current file by using the **.nx** (nextfile) request:

.nx **file**

When you select **file** with the **.nx** request, it becomes the current file.

The **nroff** formatter has a **.pi** (pipe) request that is not recognized by **troff**:

.pi **program**

The **nroff** output of the processed file is piped to **program**. No **nroff** parameters are passed to **program**, and the **.pi** request must occur in input before any output occurs.

2.8.18.2 Insertions From Standard Input

You can switch temporarily to standard input using the **.rd** (read) request. The formatter then reads from standard input until it reads two new-lines in a row (it discards the extra blank line) and returns to the input file it is processing. On the AIX PS/2 Operating System, the standard input can be keyboard, a pipe, or a file. Normally, this feature is used for insertions in form-letter style documentation, but you can use it when you want to cause a pause for input (for instance, so you can change the printer setup). The read request is of the form:

```
.rd prompt
```

By default, **prompt** is the ASCII **BEL** character, which causes most terminals to generate an audible signal to the operator.

If the formatter is taking insertions from the terminal keyboard while output is being printed on the terminal, the **-q** flag, when used in the formatter command-line, turns off the echoing of keyboard input and prompts only with BEL.

Note: Regular input and insertion input cannot simultaneously come from the standard input.

You can, for example, prepare multiple copies of a form letter by entering the insertions for all the copies in one file to be used as the standard input causing the file containing the letter to summon itself using a **.nx** request (see "File Switching and Piping" in topic 2.8.18.1). The process is ultimately ended by an **.ex** (exit) request in the insertion file.

The **.ex** (exit) request causes an exit from the formatter. **.ex** is used without parameters and it stops all text processing as though all input has ended.

2.8.18.3 Messages to Standard Output

Various messages are written to standard message output for the AIX PS/2 Operating System. The standard message output is different from the standard output, where **nroff** formatted output goes. By default, both are written onto the user's terminal, but they can be independently redirected using the requests that follow. Error messages also go to standard message output, as does the prompt generated by the **.rd** request.

Subtopics
2.8.18.3.1 Deliberate Messages
2.8.18.3.2 Error Messages

2.8.18.3.1 Deliberate Messages

You can imbed a message in the input that is stored in copy mode during formatting and cause it to be sent to the terminal at an appropriate time during output using the **.tm** (transmit) request:

```
.tm [string]
```

where **string** is output to the user terminal. Any leading blank characters in **string** are ignored. If not specified, **string** is a new-line character.

You can examine some characteristics of your document by including the **.pm** (print macros) request:

```
.pm [t]
```

The names and sizes of all defined macros and escape requests are displayed on the user terminal. The **t** parameter causes only the total of the sizes is displayed. The sizes are given in 128-character blocks.

2.8.18.3.2 Error Messages

Various error conditions can occur during the operation of the formatter. Some 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), and ***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 excess is discarded, and in **troff**, the affected word or line is marked at the point of truncation with a pointer marker. The formatter continues processing, if possible, because output useful for debugging may be produced. If a serious error occurs, processing stops and an appropriate error message is output. Examples of conditions that end processing are the inability of the formatter to create, read, or write files and the exceeding of certain internal formatter limits that make future output unlikely to be useful.

2.8.18.4 Flushing an Output Buffer

The **.fl** (flush) request causes any text stored in the output buffer to print; it is useful in interactive debugging. **.fl** is used without parameters.

2.9 Chapter 9. Formatting Mathematics

Subtopics

2.9.1 About This Chapter

The **eqn** command is designed so that you can control the mathematics formatting process. It preprocesses, or filters, an input file and converts **eqn** keywords into a series of requests that **troff** processes to give a consistent and correct appearance to typeset mathematics. Macro packages, such as viewgraph and memorandum macros, can use **eqn** calls and imbed equations in lines of text or in displays within text.

Mathematical text uses a mixture of Roman, italic, Greek, and special characters, arithmetic and logic operators, and mixed type sizes; and it has typographical conventions that are quite different from those of ordinary text. Moreover, mathematical expressions have a two-dimensional structure that is more complex to prepare than the essentially linear structure of normal text. Mathematical formulae require line-drawing and built-up characters like braces and radicals. They present a spectrum of character-positioning problems.

The **eqn** formatter provides all of the facilities you need for doing mathematics formatting, such as arbitrary horizontal and vertical motions, line-drawing, and type size changing. The **neqn** command understands most **eqn** keywords but is more limited because it cannot support different type sizes, fonts, and the assembly of large formula structures through use of a special graphics character set. These are limitations imposed by the **nroff** formatter it uses and the output devices that **nroff** drives. **neqn** is generally used to examine output before processing the input file through **eqn** and **troff**, but for some purposes **neqn** output is adequate. If you are not using **troff** and phototypesetting the output, you will find the usability of the equation formatter to be limited. Be aware that while **eqn** assumes that text other than keywords is printed in italic font, **neqn** always assumes Roman font unless you specify otherwise.

Because **eqn** and **neqn** use essentially the same keywords, generally when **eqn** is mentioned in this section, **neqn** is meant also. Similarly, when **troff** is mentioned, **nroff** is to be inferred when **neqn** is used. Where there are specific differences in applicability of a keyword, the text makes it clear. When the term ***equation formatter*** is used in this guide, both **eqn** and **neqn** are meant.

Some examples shown in this section (and elsewhere in this manual) are simulations of **eqn** and **troff** output, while others are actual results. Actual output appearance may vary slightly from the examples shown. Differences in typesetting hardware and in selected typefaces can result in differences in printout appearance.

The equation formatter does not examine the mathematics formulae it processes for accuracy or validity. In particular, mathematical symbols, like +, -, ¦, parentheses and so on, do not have special meanings.

The normal use of an equation formatter is to prepare a document with both mathematics and ordinary text interspersed. **eqn** accomplishes this by formatting mathematics and passing text on to the typesetter program that prepares the body of text.

You cannot use the **eqn** or **neqn** programs to format Japanese text as these programs do not support multibyte characters.

2.9.2 Command Lines

To use a mathematics preprocessor, you filter the text (normally as a file or files) containing the appropriate preprocessor delimiters and keywords through the preprocessor and then through a formatting program.

Thus to use **eqn**, you enter:

```
eqn file... | troff
```

Options, if any, follow the formatter part of the command. For example, you can use the memorandum macro package:

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

To use **neqn**, you enter:

```
neqn file... | nroff
```

neqn is used to drive line printers and printing devices like the Teletype Model 37. The above command is appropriate for a Model 37 terminal.

Both **eqn** and **neqn** can be used with the **tbl** (Chapter 10, "Formatting Tables") for setting tables that contain mathematics. Use **tbl** in the command line before an equation filter:

```
tbl file... | eqn | troff
```

or:

```
tbl file... | neqn | nroff
```

Command lines that are usable for most printers are:

```
tbl -TX file ... | neqn | nroff -Tlp | col -x
```

or:

```
tbl -TX file ... | neqn | nroff -Tlp -mm | col -x
```

or:

```
mm -t -e -T745 file ...
```

The last command line shown here is the equivalent of the command line that precedes it. Other ways of calling **eqn**, **tbl**, **troff**, and so on, are described in Chapter 7, "Memorandum Macros."

Subtopics
2.9.2.1 Equation Formatting Language
2.9.2.2 Equation Formatting Syntax

2.9.2.1 Equation Formatting Language

You can picture an equation as a set of boxes, bunched together in various ways. For example, something with a subscript is a box followed by another box, moved downward and shrunk by an appropriate amount. A fraction is just a box centered above another box, at the right height, with a line of appropriate length drawn between them. **eqn** is a language for specifying these boxes and for creating equations out of these boxes.

A subset of the grammar for **eqn** follows. Words in lowercase letters are literal strings (with the exception of **text**); words in uppercase letters are syntactic categories. The **|** (vertical bar) indicates an alternative; the **[** and **]** (brackets) indicate optional material. **text** represents a string of nonblank characters or any string inside double quotes. For example:

```
EQN  : BOX | EQN BOX

BOX  : TEXT
     | { EQN }
     | BOX over BOX
     | sqrt BOX
     | [ l | c | r ]pile { LIST }
     | BOX sub BOX | BOX sup BOX
     | left TEXT EQN [ right TEXT ]
     | BOX [ from BOX ] [ to BOX ]
     | size TEXT BOX
     | [roman | bold | italic] BOX
     | BOX [hat | bar | dot | dotdot | tilde]
     | define TEXT TEXT

LIST : EQN | LIST above EQN

TEXT : text
```

The observation that **something** can be replaced by a more complicated **something** in braces is implicit in the constructions:

```
EQN : BOX | EQN BOX
BOX : TEXT | { EQN } | ...
```

Anywhere **TEXT** can be used, any legal box construction can be used.

The **eqn** grammar appears ambiguous, but it is supplemented with a few rules that describe the precedence and associativity of operators. In particular, it specifies (more or less arbitrarily) that **over** associates to the left. On the other hand, **sub** and **sup** bind to the right because this is closer to standard mathematical practice. That is, **eqn** assumes **x(a(b**)) is **x**((**a(b**))), not (**x(a**))(**b**).

The precedence rules resolve the ambiguity in a construction like:

```
a sup 2 over b
```

We define **sup** to have a higher precedence than **over**, so this is parsed as:

```
a sup 2 over b
```

You can force a particular interpretation by placing braces around expressions.

The **eqn** preprocessor scans the input text. Any time it recognizes an **eqn** construct, it (potentially) creates **troff** commands as output. Thus, whenever **eqn** finds a **text** (a string of nonblank characters or a quoted string), **eqn** has found a production of:

```
TEXT     : text
```

The processing of this is simple. **eqn** generates a local name for the string, then passes the name and the string through to **troff**. **troff** stores the string. **eqn** saves only the name, the string's height, and its baseline.

As another example, the interpretation associated with the construction:

```
BOX     : BOX over BOX
```

is read (by **troff**) as:

```
Width of output box = slightly more than largest input width
Height of output box = slightly more than sum of input heights
Base of output box = slightly more than height of bottom input box
String describing output box =
        move down;
        move right enough to center bottom box;
        draw bottom box (copy string for bottom box);
        move up; move left enough to center top box;
        draw top box, (copy string for top box);
        move down and left; draw line full width;
        return to proper base line.
```

Most of the other constructions have equally simple semantic actions. Picturing the output as a set of properly placed boxes makes a valid sequence of positioning keywords apparent. The main difficulty is in finding the right dimensions to use for aesthetically pleasing positioning.

2.9.2.2 Equation Formatting Syntax

The **eqn** syntax is designed to be usable by people who do not know mathematics or typesetting:

Normal mathematical conventions about operator precedence an parentheses are not used. The equation formatter does not assume that parentheses and brackets are always balanced as opening and closing pairs. It does not substitute one form of an equation for another; it will compose the construction you call but does not impose stylistic or consistency conventions on your output.

Standard things happen automatically. **troff** subscripts and superscripts are automatically printed in an appropriately smaller size, with no additional keywords. Fraction bars are made the right length and positioned correctly, and so on. (**neqn/nroff** adjustments are more limited.)

You can override default actions for special purposes

The **eqn** preprocessor keywords convert cleanly to formatter requests so you can format intermingled mathematics and text. When you prepare text, you enter mathematical expressions as part of the text, but mark them by equation delimiters. **eqn** reads this input and treats strings that are not marked as equations as comments, passing them through to **troff** unprocessed. At the same time, **eqn** converts equation input into the necessary **troff** requests. The resulting output passes directly to **troff** where text and further requests are processed.

2.9.3 Keywords

The following keywords are the entire function vocabulary recognized by **eqn**.

Keywork	Discussed in Topic:	Keyword	Discussed in Topic:
above	2.9.17	lpile	2.9.17
back	2.9.21	mark	2.9.15
bar	2.9.10.4	matrix	2.9.18
bold	2.9.11	ndefine	2.9.8.2
ccol	2.9.18	over	2.9.12
col	2.9.18	pile	2.9.17
cpile	2.9.17	rcol	2.9.18
define	2.9.8.2	right	2.9.19
delim	2.9.5	roman	2.9.11
dot	2.9.10.4	rpile	2.9.17
dotdot	2.9.10.4	size	2.9.11
down	2.9.21	sqrt	2.9.13
dyad	2.9.10.4	sub	2.9.14
fat	2.9.11	sup	2.9.11
font	2.9.11	tdefine	2.9.8.2
from	2.9.20	tilde	2.9.10.4
fwd	2.9.21	to	2.9.20
gfont	2.9.11	under	2.9.10.4
gsize	2.9.11	up	2.9.21
hat	2.9.10.4	vec	2.9.10.4
italic	2.9.11	~	2.9.10.1
lcol	2.9.18	{	2.9.16
left	2.9.19	}	2.9.16
lineup	2.9.15	(^)	2.9.10.1

eqn also recognizes the use of tabs (page 2.9.10.1) and new-line (page 2.9.10.1) characters as delimiters.

2.9.4 Equation Display Delimiters

The **eqn** preprocessor reads intermixed text and equations and passes its output to the formatter. The formatter uses the macros **.EQ** and **.EN** to delimit an equation, which is kept intact by the formatter, for example:

```
Some text.
.EQ
x=y+z+1
.EN
Some more text.
```

This results in:

Some text. **x=y+z**+1 Some more text.

The equation is converted to instructions that tell the formatter the font, typeface, character, and location to use for each part of the equation. **eqn** routinely provides some text specifications, such as changes in typeface for equations. Variables are made italic, operators and digits become Roman, and normal spacings between letters and operators are adjusted to give a satisfactory appearance to equations and formulae. The **neqn** output lacks the ability to perform all **eqn** actions but still produces equation output, ignoring keywords it cannot perform or compensate for.

The formatter treats an equation as a single line as far as formatter requests acting on it and reserves necessary vertical space to present an equation intact. Ordinarily, the formatter simply inserts the equation in the text inline as though it were any text, but there are manipulations you can perform on your equations, such as numbering them or aligning them in columns. You can, for instance, use the formatter request **.ce** to center an equation:

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

You have to take care of things like centering, numbering, and so on yourself; the easiest way is to use the memorandum macros, which provide useful equation formatting aids such as centering, indentation, left margin justification, and equation numbering. Different macro packages use **eqn** in different ways. The memorandum macros, however, let you imbed equations in displays and label them with a string parameter to the **.EQ** macro (see "Equations" in topic 2.7.4.11.4). You should examine the way that equations are displayed by the macro package you are using.

2.9.5 Inline Equation Delimiters

In a mathematical document, you must follow mathematical conventions not just in equation displays but also in the body of the text; for example, you make variable names like **x** italic. You can do this by surrounding the appropriate parts with **.EQ** and **.EN**, but then you have to repeatedly enter **.EQ** and **.EN**.

eqn provides a shorthand for brief inline expressions. Using the **delim** keyword, you can define two characters to mark the left and right ends of an inline equation, then insert expressions in the middle of text lines. The opening and closing characters can be the same; once **eqn** has set the characters, **troff** continues to look for the delimiters in pairs. To set both the left and right characters as **eqn** delimiters to dollar signs, for example, add three lines at the start of the document:

```
.EQ
delim @@
.EN
```

Having done this, you can have **troff** input like:

```
Let @alpha sub i@ be the primary variable, and let @beta@ be zero.
Then we can show that @x sub 1@ is @>=0@.
```

This works as you might expect. Spaces, new-lines, and so on are significant in the text but not in the equation part itself:

Let &alpha.[**i**] be the primary variable, and let ß be zero. Then we can show that **x[i**] is =**0**.

Multiple equations can occur in a single input line. **troff** leaves enough room before and after a line that contains inline expressions so that inline elements do not interfere with the surrounding lines.

To turn off the delimiters:

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

Notes:

1. Do *not* use braces, tildes, circumflexes, backslashes, or double quotes as delimiters. Since they have special meanings to **eqn** (and also therefore to the formatter), the result may be very different from what you intend.

2. In-line font changes must be closed before inline equations are encountered to prevent the loss of font change keywords.

2.9.6 Delimiting Input

Input is free-form; you enter spaces and new-lines to separate pieces of the input, but **eqn** does not use them to create space in the output. Thus:

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

also gives **x=y+z**+1. The free-form input feature lets you input an expression as short lines. Spaces and new-lines within mathematical expression delimiters are used to separate elements and are discarded by **eqn**. (Normal text is left alone.) Thus, all of the following, when between **.EQ** and **.EN** produce **x=y+z**:

```
x=y+z

x = y + z

x  =  y
  + z
```

In the following example, **eqn** uses space delimiters and automatically identifies special terms:

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

This is printed as:

x=2&pi.&integral.sin(&omega.**t)dt**

The characters and phrases that are not recognized as special (x,t, and dt) are printed in italic.

You can force a selected amount of blank space into the output by using characters that represent various amounts of forced blank space (see "Symbols, Special Names, and Special Characters" in topic 2.9.10).

2.9.7 Protected Input

Input that is set entirely within quotes (**"..."**) is not subject to any of the font changes and spacing adjustments normally done by **eqn**. This lets you do your own spacing and adjusting:

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

This results in:

sin(x)+sin(**x**)

You can also use quotes to get braces and other **eqn** key characters and keywords printed as literal strings:

```
  "{ size alpha }"
```

yields:

{ size alpha }

The instruction:

```
  roman "{ size alpha }"
```

causes:

{ size alpha }

You can use the construction **""** (pair of quote marks) as a place-holder when grammatically **eqn** needs something, but you do not actually want anything in your output. For example, to make (2)He, you cannot just enter **sup 2 roman He** because a **sup** has to be a superscript *to* something. You can instead enter:

```
  "" sup 2 roman He
```

To get a literal quote, use a **\"** sequence. (Thus, you cannot use **\"** for comments in **eqn**.) You can use **troff** special characters like **\(ci** unquoted, but more complicated things like horizontal and vertical motions with **\h** and **\v** should always be quoted.

2.9.8 Defining Strings and Macros

The **eqn** preprocessor provides a means of defining characters and character combinations that can then be used as string names or macros.

Subtopics
2.9.8.1 Strings
2.9.8.2 Macros

2.9.8.1 Strings

The **eqn** preprocessor lets you name a frequently used string of characters and thereafter just give the string name instead of the whole string. For example, if the sequence:

```
x sub i sub 1 + y sub i sub 1
```

appears repeatedly throughout a paper, you can save time and labor by defining it like this:

```
define xy 'x sub i sub 1 + y sub i sub 1'
```

This defines **xy** as a name for the string of characters between the apostrophes in the definition.

Note: You can use any single character (not only a quote mark) to delimit the definition, as long as it does not appear inside the definition. The same character that is at the front of the definition always closes the definition. If you precede the string with an open single quote (`) mark then you will end it with the same mark, not a close single quote or apostrophe (') mark.

You can use the string name **xy** defined as above in constructions like:

```
.EQ
f(x) = xy ...
.EN
```

The **eqn** preprocessor replaces the string name **xy** with the string it is defined as. Be careful to leave spaces or their equivalent around the name when you actually use it so that **eqn** will identify it as a string name.

Although strings can include string calls, as in:

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

do not define something in terms of itself. A common error is to say:

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

This is an error, since **X** is now defined in terms of itself. You can get around the difficulty and do what you intend by entering the **X** literally:

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

2.9.8.2 Macros

You can define or redefine **eqn** keywords as macros with the **define** keyword. You can, for example, make **(^)** mean **over** by saying:

```
  define (^) ' over '
```

or redefine **over** as **(^)** with:

```
  define over ' (^) '
```

If you need to print different things on a printer and on the typesetter, you may want to define a symbol to have a different meaning in **neqn** than it has in **eqn**. You can do this with the keywords **ndefine** and **tdefine**.

A definition made with **ndefine** only takes effect if you are running **neqn** and is ignored by **eqn** and **troff**. If you use **tdefine**, the definition only applies for **eqn** and is ignored by **neqn** and **nroff**.

Macros defined with a **define** keyword apply to both **eqn** and **neqn**.

2.9.9 Associativity and Precedence

Operations group to the right, except these, which group to the left: **over**, **sqrt**, **left**, and **right**.

If you do not use braces for grouping, **eqn** will do operations in the order shown in this list.

1. Diacritic Marks and Logic Operators

   dyad
   vec
   under
   bar
   tilde
   hat
   dot
   dotdot

2. Local Motions

   fwd
   back
   down
   up

3. Font and Typeface

   fat
   roman
   italic
   bold
   size

4. Halfline Motions, Square Roots, and Fractions

   sub
   sup
   sqrt
   over

5. Summation and Similar Constructs

   from
   to

2.9.10 Symbols, Special Names, and Special Characters

The **eqn** preprocessor recognizes some mathematical symbols and names, special characters, and the Greek alphabet. If you enter:

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

this produces:

x=2&pi.&integral.sin(&omega.**t)dt**

The delimiter spaces in the preceding input tell **eqn** that **int**, **pi**, **sin** and **omega** are separate items that get special treatment. The **sin**, the numeral **2**, and the parentheses are recognized as math terms and set in Roman type; **pi** and **omega** are replaced by the Greek symbols they represent (&pi. and &omega.); and **int** becomes the integral sign (&integral.). The characters and character sequences not given special interpretation (x, t, and dt) are printed in italic (Roman in **neqn/nroff**).

Subtopics

2.9.10.1 Forced Space and Special Delimiters

You can use a tilde (~) for each required space you want to force into the output:

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

This gives:

x = y + z

You can also use a circumflex (**(^)**), which gives a space half the width of a tilde; it is useful mainly for perfecting the appearance of output equations. Tabs can also position elements of an expression. The tab stops must be set by formatter requests, though there are default tab stops (see "Tabs and Leaders" in topic 2.8.16.2.2).

The **eqn** preprocessor recognizes that a string of characters can be a special name if the string is separated from the characters on either side of it. You can do this by delimiting a special name with ordinary spaces or instead by using tabs, new-lines, tildes, or circumflexes, for example:

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

The tildes not only separate the special names like **sin** and **omega** but also add extra spaces, one space per tilde.

Special names can also be separated by braces and double quotes, which also have special meanings (see "Protected Input" in topic 2.9.7).

2.9.10.2 Mathematical Words and Symbols

Digits, parentheses, brackets, punctuation marks, and these mathematical words are output in Roman font when encountered:

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

2.9.10.3 Special Characters

These character sequences are recognized and translated as shown. Some of the characters listed cannot be printed on some printers. There is a collection of predefined character **eqn** macros you can define; see the *AIX Operating System Technical Reference*. You can also use **troff** four-character escape sequences, which **eqn** does not recognize, but passes on to **troff** to convert into a character (see Appendix A, "Escape Sequences").

Keyword	Typeset Output	Keyword	Typeset Output
>=	=	**approx**	&approx.
<=	=	**nothing**	
==	==	**cdot**	.
!=	&nesym.	**times**	¦
+-	±	**del**	&del.
->		**grad**	&del.
<-		**...**	...
<<	<<	**,...,**	,...,
>>	>>	**sum**	&sum.
inf	&infinity.	**int**	&integral.
partial	&partial.	**prod**	&Pi.
half	½	**union**	&union.
prime	'	**inter**	&intersect.

2.9.10.4 Diacritic Marks and Logic Operator Accents

Characters can be modified by following them with a keyword that **eqn** interprets as a call for a modifying mark. To get modifying marks for characters, there are several keywords:

Keyword	Description
dot	Places a pip (single dot) centered above the character.
dotdot	Places a diaeresis (umlaut or double dot) mark centered above the character.
hat	Places a circumflex (hat) mark above the character.
tilde	Places a tilde (negation) mark above the character.
vec	Places a vector (right arrow) mark above the character.
dyad	Places a dyad (double arrow) operator above the character.
bar	Places a macron (bar) mark above the character.
under	Places an underscore mark below the character.

The selected mark is automatically placed at the correct line height for the character, symbol, or construct it is placed with. The **bar** and **under** are made the right length for the entire construct; all marks are centered. **eqn** makes no assumptions about proper use of the marks.

2.9.10.5 Greek Characters

To obtain Greek letters, spell them out in whatever case you want. (Many printers cannot produce Greek characters and will only print those characters with Roman equivalents.)

Keyword	Typeset Character	Keyword	Typeset Character
DELTA	&Delta.	**iota**	&iota.
GAMMA	&Gamma.	**kappa**	&kappa.
LAMBDA	&Lambda.	**lambda**	&lambda.
OMEGA	&Omega.	**mu**	¦
PHI	&Phi.	**nu**	&nu.
PI	&Pi.	**omega**	&omega.
PSI	&Psi.	**omicron**	&omicron.
SIGMA	&Sigma.	**phi**	&phi.
THETA	&Theta.	**pi**	&pi.
UPSILON	&Upsilon.	**psi**	&psi.
XI	&Xi.	**rho**	&rho.
alpha	&alpha.	**sigma**	&sigma.
beta	ß	**tau**	&tau.
chi	&chi.	**theta**	&theta.
delta	&delta.	**upsilon**	&Upsilon.
epsilon	&epsilon.	**xi**	&xi.
eta	&eta.	**zeta**	&zeta.
gamma	&gamma.		

2.9.11 Size and Font Changes

By default, **eqn** equations are set in 10-point type, with standard mathematical conventions to determine which characters are in Roman and which are in italic. Although **eqn** uses aesthetically pleasing sizes and fonts (**neqn** does not attempt to use italics unless specifically told), the results are not always perfect or appropriate. You can change sizes and fonts using certain keywords:

Keyword	Description
size	Sets type size using the keyword **size** followed by a number, which is the type size in points (approximately 1/72 inch of height per point). Legal sizes which can follow **size** are the 15 type sizes supported by **troff**: 6, 7, 8, 9, 10, 11, 12, 14, 16, 18, 20, 22, 24, 28, and 36. You can also change the size by a given amount using a plus or minus operator; for example, you can give **size+2** to make the size 2 points bigger.
roman	Sets the font to Roman, the default font for mathematical words in **troff**; the default font in **nroff**.
italic	Sets the font to italic, the default font for equation variables and other text.
bold	Sets the font to (Roman) bold font.
fat	Causes printing (of current type size and font) in an emphasized mode by overprinting with a slight offset.

Like **sub** and **sup**, a size or font change affects only the element that follows it and then reverts to the normal situation. Thus, if you enter:

```
  bold x y
```

this results in:

x y

If you enter:

```
  fat x y
```

this results in:

x **y**

If you enter:

```
  .EQ
  size 8 bold {A~x~=~y}
  .EN
```

this produces:

A x = y

The **size** keyword is followed by a number representing a type size in points; the font is directly specified.

You can select a particular font using the keyword **font** to select any font that **troff** recognizes. You give a 1-character parameter to **font** that is a character that **troff** recognizes as a call for a font. To select Helvetica font, for example, enter:

```
font H
```

If you use fonts other than Roman, italic, and bold, the fonts may not give quite as good an appearance; **eqn** is designed to use Roman, italic, and bold as fonts.

You can place an input string in quotes to cause **eqn** to treat it as text, ignoring any keyword significance contained in it. For example:

```
.EQ
lim~ roman "sup" ~x sub n =0
.EN
```

this causes a literal **sup** to print in Roman font and ensures that the **x** does not become a superscript:

lim sup **x[n**]=0

If the equations of an entire document are to be in a nonstandard size or font, it is a nuisance to have to write out a size and font change for each equation. You can set a global size or font which thereafter affects all subsequent equations by using the keywords **gsize** or **gfont**. At the beginning of any equation, you might say, for instance:

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

This sets the type size to 16 and the font to Roman for each equation thereafter. The font can be given as a recognized type specification character or as a physical position number. The size after **gsize** can be a relative change with + or -.

Generally, you would use **gsize** and **gfont** near the beginning of a document, but they can also appear throughout a document: you can change the global font and size as often as needed. Use them within **eqn** delimiters **.EQ** and **.EN** so **eqn** will recognize them as keywords. For example, in a footnote you will typically want the size of equations to match the size of the footnote text, which is two points smaller than the main text. Do not forget to reset the global size at the end of the footnote.

2.9.12 Fractions

You can build a fraction with the word **over**:

```
  a+bd over 2c =1
```

This gives:

```
  <a+bd> over 2c = 1
```

The line is made the right length and positioned automatically. Braces can be used to make clear what goes over what:

```
  {alpha + beta} over {sin (x)}
```

This yields:

```
  alpha+beta over <sin operator (x)>
```

eqn interprets **sup** before **over**, to make it unambiguous as to what happens when there is both an **over** and a **sup** in the same expression:

```
  -b sup 2 over pi
```

This results in:

```
  -b sup 2 over pi
```

The rules which decide ***precedence*** (which operation is done first) in cases like this are summarized under "Associativity and Precedence" in topic 2.9.9. If you are in doubt, you can simply use braces to make clear what groups with what.

2.9.13 Square Roots

To draw a square root, use **sqrt**:

```
  sqrt a+b + 1 over sqrt {ax sup 2 +bx+c}
```

This is:

```
  sqrt a+b + 1 over sqrt <ax sup 2 +%bx%+%c>
```

Square roots of tall (piled) quantities look poor when typeset, because a root-sign big enough to cover the quantity is very dark and heavy:

```
  sqrt {a sup 2 over b sub 2}
```

This is:

```
  sqrt <a sup 2 over b sub 2>
```

You may prefer to write big square roots as something to the power of ½:

```
  (a sup 2 /b sub 2 ) sup half
```

This results in:

(**a**(2)/**b**[2])(½)

2.9.14 Superscripts and Subscripts

Subscripts and superscripts are set with the keywords **sub** and **sup**:

```
  x sup 2 + y sub k
```

This gives:

x(2)+**y[k**]

eqn takes care of all size changes and vertical motions needed to make the output look right. The words **sub** and **sup** must be delimited by spaces; **xsub2** gives you **xsub**2 instead of **x**[2]. Do not forget to leave a space (or tilde, and so on) to mark the end of subscript or superscript. A common error is to say something like:

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

This results in:

y=(**x**(2)+1)

instead of the intended:

y=(**x**(2))+1

The equation formatter automatically returns to the proper baseline following the superscript. Multiple levels of subscripts and superscripts can accumulate:

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

yields:

x(y(z))

and:

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

yields:

x(y[z])

Note: There is an exception to cumulative superscripts and subscripts. **eqn** recognizes the construct **something sub something sup something** as a special case, so **x sub i sup 2** is formatted as **x[i**](2) instead of **x[i**]2. Other than this case, **sub** and **sup** group to the right.

2.9.15 Arbitrary Equation Alignment

Sometimes it is necessary to line up a series of equations at some columnar position, aligning for instance on an equals sign. You can do this with keywords, **mark** and **lineup**.

The keyword **mark** can appear at any place in an equation and can be used only once in the equation. **eqn** remembers the ***column*** or horizontal position where it appears. Successive equation elements can contain one occurrence of the word **lineup**. The place where **lineup** appears is forced to line up with the place marked by the previous **mark** if at all possible. Thus, for example, you can say:

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

to produce:

x+y=z
x=1

Note: The **mark** and **lineup** keywords do not work with centered equations. Also, **eqn** does not look ahead of **mark** to see whether the **lineup** point will fit. The following example does not work, because there is not room for **x+y** to be placed to the left of the place **eqn** has already set as the alignment mark.

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

2.9.16 Explicit Grouping

You can form complicated expressions with primitives. For example:

```
.EQ
{partial sup 2 f} over {partial x sup 2} =
 x sup 2 over a sup 2 + y sup 2 over b sup 2
.EN
```

produces:

```
<partial sup 2 f> over <partial x sup 2> =
<x sup 2 over a sup 2> + <y sup 2 over b sup 2>
```

You can always use {...} (braces) to explicitly group objects together; in the above case they indicate what goes over what on the left side of the expression.

The **eqn** keyword language defines the precedence of **sup** to be higher than that of **over**, so you do not need braces to get the correct association on the right side. However, you can always use braces when you are in doubt about precedence. If a construct can appear in some context, then any expression in braces can also occur in the context.

Braces can be used to prevent a blank (or other character) from delimiting some element at the wrong place, forcing all the input between the braces to be acted on together. For example, if a subscript or superscript is something that has to be typed with blanks in it, you can use the braces to mark the beginning and end of subscript or superscript:

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

This yields:

```
e sup <i omega t>
```

The general rule is that anywhere you could use some single element, you can use an arbitrarily complex element if you enclose it in braces. **eqn** automatically handles all the details of positioning and size. Compare these examples:

```
x sub {i sub 1} sup 2

x sub i sub 1 sup 2
```

These result in:

```
x % sup 2 sub <i % sub 1>

x % sub <i % sub 1 sup 2>
```

You can use braces within braces:

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

This results in:

```
e sup <i pi sup <rho + 1>>
```

In all cases, make sure you have matching left and right braces. Leaving

one out or adding an extra will confuse **eqn**.

Occasionally, you may want to explicitly print braces. To do this, enclose them in double quotes, like **"{"**. Using quote marks is discussed in more detail under "Protected Input" in topic 2.9.7.

2.9.17 Equation Internal Vertical Alignment

The **eqn** preprocessor has the ability to make orderly vertical piles of formula elements, stacking them and aligning them to the left, center, or right of the pile.

The **pile** keyword stacks construction elements. For example:

```
A ~=~ left [
 pile {a above b above c}
 ~~ pile {x above y above z}
 right ]
```

results in:

```
A = left lbracket a above b above c
% above % above %
x above y above z right rbracket
```

The elements in the pile (there can be as many as you want) are centered one above another, at the correct separation for most purposes.

The keyword **above** is used to stack the elements and enclosures are used around the entire pile of elements. The elements in a stack can be as complex as needed. They can even contain more piles.

There are three other stacking keywords:

lpile Makes a pile with the elements left-aligned.
rpile Makes a right-aligned pile.
cpile Makes a centered pile, much like **pile**.

The vertical spacing between the elements is somewhat larger for **lpile**, **rpile**, and **cpile** than for an ordinary **pile** (see the **pile** example in topic 2.9.19).

2.9.18 Matrixes

You can make equation matrixes using **eqn** and the **ccol** keyword. For example:

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

results in:

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

The elements of the columns are listed just as they are for a pile, with each construct separated by the word **above**. You can also use **lcol** or **rcol** to align columns to the left or to the right. You can make and separately align as many columns as you like.

The **col** keyword builds a matrix with less vertical separation than **ccol** (or **lcol** or **rcol**).

A reason for using a matrix instead of two adjacent piles is that if the elements in the piles do not all have the same height, they will not stack up evenly. A **matrix** call results in vertical and horizontal alignment of elements, because **eqn** looks at the entire equation structure before deciding what spacing to use.

Note: Each column of a matrix **must** have the same number of elements in it. Of course, you can create empty elements ("Protected Input" in topic 2.9.7).

2.9.19 Enclosing Elements

To enclose elements in big [brackets], {braces}, (parentheses), and |bars|, use the **left** and **right** enclosure keywords:

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

This results in:

```
left lbrace <a over b> + 1 right rbrace =
left lparen c over d right rparen +
left lbracket e right rbracket
```

The resulting brackets are made big enough to cover whatever they enclose. Other characters such as the **floor** and **ceiling** characters can be used in the same way. For example:

```
left floor x over y right floor
< left ceiling a over b right ceiling
```

produces:

```
left lbracket <x over y> right rbracket lt
left lbracket <a over b> right rbracket
```

The enclosure keywords **left** and **right** are conveniently used as pairs. However, you can omit the **right** part of an enclosure pair. If the **right** part is omitted, put braces around the thing you want the left bracket to encompass. Otherwise, the resulting enclosure brackets can be too large.

You can also use the **right** keyword and use an empty **left** part if you do not want a left enclosure to appear. You cannot have a **right** without a preceding **left** keyword. You will have to create an empty **left**. For example:

```
left "" {a over b} right ]
```

results in:

```
left ' ' a over b right rbracket
```

Note: There are some limitations in using enclosures to achieve eye-pleasing output. Braces are often taller than brackets and parentheses, because they are made up of three, five, seven, ..., parts while brackets can be made up of two or more parts. Also, the formatter cannot assemble large constant-radius curves for large parentheses, so they only curve toward the tips.

You can adjust the height of braces, brackets, parentheses, and vertical bars using the keywords **left** and **right**. For example:

```
.EQ
left [ x+y over 2a right ](~)=(~)1
.EN
```

gives:

```
left lbracket <x + y> over 2a right rbracket = 1
```

A **left** need not have a corresponding **right**, as in the next example. You can use any characters following **left** and **right**, but generally only certain parentheses and bars are meaningful.

Big brackets and similar constructs are often used with another keyword called **piles**, which makes vertical groupings of objects (see **piles**, in topic 2.9.17). For example:

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

produces:

```
'sign' operator (x) == left lbrace 1 rabove 0 rabove -1
% 'if' above 'if' above 'if' %
% above % above %
x above x above x
% above % above %
gt above = above lt
% above % above %
0 above 0 above 0
```

The construction **left {** makes a left brace tall enough to enclose the **rpile {...}** construction, and thus, the **lpile {...}** construction.

2.9.20 Summations, Integrals, and Similar Constructs

Summations, integrals, and similar constructions are called by using a keyword for the symbol and the keywords **from** and **to** to specify the limits. For instance:

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

produces:

```
  sum from i=0 to infinity of x sup i
```

The **from** and **to** segments are both optional; but if both are used, they have to occur in that order. Centering and making the &sum. large enough and the limits smaller are automatic. The most commonly used symbols in these constructs are:

Integral &integral. is called by using the keyword **int**.

Intersection &intersect. is called by using the keyword **inter**.

Product &product is called by using the keyword **prod**.

Summation &sum. is called by using the keyword **sum**.

Union &union. is called by using the keyword **union**.

The element before the **from/to** can be anything. For example:

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

is:

lim **x[n]**=0
((**n** &infinity.))

Notice in the latter example that braces indicate where the **n &infinity.** begins and ends. No braces were necessary in the former example for the **i=0**, because it did not contain space characters. The braces are never unacceptable, and if the **from** and **to** segments have any space characters, you must use braces around them. For example:

```
  .EQ
  lim from {x -> pi /2} ( tan(~)x) = inf
  .EN
```

gives:

lim (tan **x**)=(&infinity.)
(**x** &pi./2)

Again, the braces indicate just what goes into the **from** portion of the equation.

2.9.21 Local Motions

Although **eqn** automatically positions elements for printing, it may not position things appropriately for all situations. You can adjust the positioning of printed elements.

You can insert small forced horizontal spaces with the tilde and the circumflex (see "Forced Space and Special Delimiters" in topic 2.9.10.1). You can also use the keywords **back num** and **fwd num** to move elements by small amounts horizontally (where **num** is the amount in 1/100's of an em to move). For example:

```
back 50
```

is a call to place the next character (or element) back half the width of the letter **m** in the current font.

You can similarly move things up or down with **up num** and **down num**. As with **sub** or **sup**, the local motions affect the next element in the input; this can be a complex element if it is enclosed in braces.

2.9.22 Examples

The following is an example of a long equation, broken into three parts:

Subtopics
2.9.22.1 Input:
2.9.22.2 Output:
2.9.22.3 Input:
2.9.22.4 Output:
2.9.22.5 Input:
2.9.22.6 Output:

2.9.22.1 Input:

```
.EQ
G(z)~mark =~ e sup {ln ~ G(z)}
 ~=~ exp left (
 sum from k>=1 {S sub k z sup k} over k right )
 ~=~ prod from k>=1 e sup {S sub k z sup k /k}
.EN
```

2.9.22.2 Output:

```
G(z) % = % e % sup <lnG(z)> % = % exp left lparen sum from <k ge 1>
% <S sub k z sup k> over k right rparen % =
% product from <k ge 1> of % e sup <<S sub k> % <z sup k> <l> <k>>
```

2.9.22.3 Input:

```
.EQ
lineup = left ( 1 + S sub 1 z +
 {S sub 1 sup 2 z sup 2} over 2! + ... right )
 left ( 1+ { S sub 2 z sup 2 } over 2
 + {S sub 2 sup 2 z sup 4 } over { 2 sup 2 cdot 2! }
 + ... right ) ...
.EN
```

2.9.22.4 Output:

```
= left lparen 1 + % <S sub 1 z> +
<S sup 2 sub 1 % z sup 2> over 2! + ellipsis right rparen
left lparen 1 + <S sub 2 % z sup 2> over 2 +
<S sup 2 sub 2 % z sup 4> over <2 sup 2. % 2!> +
ellipsis right rparen ellipsis
```

2.9.22.5 Input:

```
.EQ
lineup = sum from m>=0 left (
sum from
pile { k sub 1 ,k sub 2 ,..., k sub m >=0
above
k sub 1 +2k sub 2 + ... +mk sub m =m}
{ S sub 1 sup {k sub 1} } over {1 sup k sub 1 k sub 1 ! } ~
{ S sub 2 sup {k sub 2} } over {2 sup k sub 2 k sub 2 ! } ~
...
{ S sub m sup {k sub m} } over {m sup k sub m k sub m ! }
right ) z sup m
.EN
```

2.9.22.6 Output:

```
= sum from <m ge 0> % left lparen
sum from <k sub 1 % , k sub 2 % , ellipsis , k sub m ge 0>
from <k sub 1 + 2k sub 2 + ellipsis + mk sub m = m>
% <S sub 1 sup <k sub 1>> over <1 sup <k sub 1> % k sub 1 !>
% <S sup <k sub 2> sub 2> over <2 sup <k sub 2> % k sub 2 !>
ellipsis
% <S sup <k sub m> sub m> over <m sup <k sub m> % k sub m !>
right rparen % z sup m
```

2.9.23 Troubleshooting

The most common problem in using **eqn** seems to be the blank used as a delimiter; even experienced users sometimes insert blanks where they should not and omit them when they are needed. A common instance is typing:

```
f(x sub i)
```

which produces:

```
f lparen x sub <i rparen>
```

instead of the intended:

```
f lparen x sub i rparen
```

If you define some delimiters, it is easy to leave one out of the input. This causes problems that are hard to identify. The program **checkeq** (see *AIX Operating System Commands Reference*) lets you check for misplaced or missing delimiters and similar troubles.

If you make a mistake in an equation, such as leaving out or using one too many braces or having a **sup** with nothing before it, **eqn** displays the message:

```
syntax error between lines xnum and ynum, file filename
```

where **xnum** and **ynum** are the lines between which the trouble occurred, and **filename** is the name of the file in which the lines are stored. The line numbers are only approximate, and the trouble may have occurred nearby but outside the lines given. Also, the trouble may have occurred earlier but not created a message-generating conflict until the area indicated.

If you want to check a document before actually printing it, enter the command line:

```
eqn files >\dev\null
```

This discards the output but displays the messages.

Inline equation size is limited by the capacity of an internal buffer in the formatter. If you get a **word overflow** message, you have exceeded this limit. If you print the equation as a displayed equation rather than an inline equation, this message will usually go away because a larger buffer is used. The **line overflow** message indicates you have exceeded the larger buffer. To solve this problem, break the equation into two (or more) smaller ones placed consecutively. (This is the also the solution for printed equations that are too wide for the paper.) **eqn** does not break equations by itself; you must break long equations across multiple lines.

eqn expects **troff** to use Times Roman mounted on position 1, Times Italic on position 2, Times Bold on position 3, and the Bell Laboratories Special Font on position 4 of the phototypesetter. If another font is mounted on one of these positions, the results may not look good.

2.10 Chapter 10. Formatting Tables

Subtopics

2.10.1 About This Chapter

This chapter describes **tbl**, a document formatting preprocessor for **troff** or **nroff** that enables you to specify and enter tables. Tables are made up of columns that can be independently centered, right-adjusted, left-adjusted, or aligned by decimal points. Headings can be placed over single columns or groups of columns. A table entry can contain equations text. Horizontal or vertical lines can be drawn in the table, and any table or element can be enclosed in a box. For example:

1970 Federal Budget Transfers
(In billions of dollars)

State	Taxes Collected	Money Spent	Net
New York	22.91	21.35	-1.56
New Jersey	8.33	6.96	-1.37
Connecticut	4.12	3.10	-1.02
Maine	0.74	0.67	-0.13
California	22.29	22.42	+0.13
New Mexico	0.70	1.49	+0.79
Georgia	3.30	4.28	+0.98
Mississippi	1.15	2.32	+1.17
Texas	9.33	11.13	+1.80

Note: **nroff** can approximate **tbl** output on the IBM Personal Computer Printer and the IBM Personal Computer Graphics Printer. Some features described in this chapter (for example, double lines) are not available with these printers. However, piping the output of **nroff** through **col** and using the -TX flag to **tbl** will improve the quality of the printer output.

tbl turns a simple description of a table into a **troff** or **nroff** program (list of requests) that formats the table. It isolates the portion of input that it recognizes as table data and leaves the remainder for other programs to process. You can use **tbl** with the equation formatting program **eqn** and various layout macro packages such as **mm**; **tbl**, however, does not duplicate their functions. Throughout the rest of this section, where **troff** is used, understand it to mean **troff** or **nroff**.

The input to **tbl** is text for a document, with data for tables preceded by a **.TS** (***table start***) macro and followed by a **.TE** (***table end***) macro. **tbl** processes table data, generating **troff** formatting requests. **tbl** leaves the **.TS** and **.TE** lines delimiting the table, so **troff** layout macros can use these lines to locate and block out space for tables as needed. In particular, any parameters on the **.TS** or **.TE** lines are passed to the output without being used by **tbl** and may be used by document layout macros. The format of the input is as follows:

```
optional text
.TS
table data
.TE
optional text
.TS
table data
.TE
optional text
. .
```

Each table is independent, and must contain formatting information followed by the data to be entered in the table. The formatting information, which describes the individual columns and rows of the table, may be preceded by a few options that affect the entire table. The format of data for each table is as follows:

```
.TS
options;
format.
data
.TE
```

A detailed description of options and formatting commands follows.

You cannot use the **tbl** program to format Japanese text as this program does not support multibyte characters.

2.10.2 Input Requests

As indicated above, a table contains:

1. A command to start the table

2. Global options

3. Format section describing the layout of the table entries

4. Data to be set in a table

5. A command to end the table.

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

Subtopics
2.10.2.1 Delimiting a Table
2.10.2.2 Table Global Parameters
2.10.2.3 Table Format Parameters
2.10.2.4 Table Data

2.10.2.1 Delimiting a Table

Enter the **.TS** command to start a table. End the table with the command **.TE**.

The **tbl** preprocessor interprets input data between the delimiters into data and requests to the formatter program, either **troff** or **nroff**. It passes through unchanged any input that is outside the delimiters.

2.10.2.2 Table Global Parameters

There may be a single line (only) of global options affecting the whole table. This optional line must follow the **.TS** line immediately, may contain only a list of keywords separated by spaces, tabs, or commas, and must be terminated by a semicolon. The allowable keywords are:

center	Centers the table (default is left-adjust).
expand	Makes the table as wide as the current line length.
box	Encloses the table in a box.
allbox	Encloses each item in the table in a box.
doublebox	Encloses the table in two boxes.
tab (x)	Uses character **x** instead of the ASCII **TAB** character to separate data items.
linesize (num)	Sets lines or rules in **num**-point type.
delim (xy)	Uses **x** and **y** as **eqn** (or **neqn**) delimiters.

The **tbl** program tries to keep boxed tables on one page by issuing ***need requests*** (**.ne**), calculated from the number of lines in the tables. If there are spacing requests embedded in the input, the **.ne** requests generated by **tbl** may be inaccurate. You can use normal **troff** procedures, such as display macros, to allocate space in that case. If you must must have a multiple-page boxed table, you should use macros designed for this purpose (see **.T&** command, in topic 2.10.2.3).

2.10.2.3 Table Format Parameters

The format section of the table specifies the layout of table columns. Each line of data in this section corresponds to a single line of the table except that the last line corresponds to all following lines up to the next .T&, if any (see topic 2.10.2.3), and each line contains a keyletter for each column of the table. It is good practice to separate the keyletters for each column by spaces or tabs to make them easier to read. Each keyletter is one of the following:

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, to be aligned with other numerical entries in the column so the decimal points of the numeric entries line up. A column location for the decimal point is determined. A non-printing **\&** can be used to mark the place of the decimal point. If the entry does not contain a **\&**, 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 (ones) digit. Otherwise, the item is centered in the column.
A or **a** Indicates an alphabetical sub-column; all corresponding entries are left-aligned and positioned so that the widest entry is centered in the column (see example in topic 2.10.4.13).
S or **s** Indicates a ***spanned*** entry, where the entry from the previous column continues across this column. A spanned entry is not allowed in the left column of the table.
^ Indicates a ***vertically extended*** heading, where the entry from the previous row continues through this row. A vertically extended heading is not allowed in the first row of a table.

In the example below, the items shown at the left will be aligned (in a numerical column) as shown on the right:

Input	**Output**
13	**13**
4.2	**4.2**
26.4.12	**26.4.12**
abc	**abc**
abc\&	**abc**
43\&3.22	**433.22**
749.12	**749.12**

Notes:

1. If numerical data is used in the same column with wider **L** or **R** type table entries, the widest number is centered relative to the wider **L** or **R** items. Alignment within the numerical items is preserved. This is similar to the format of **A** type data, as explained above. However, alphabetic sub-columns, requested by a keyletter, are always slightly indented relative to **L** items; if necessary, the column width is increased to force it; this is not true for numeric type entries.

2. **N** and **a** keyletters should not be used in the same column. (There is only one number register for each column to set that indicates whether the column is alphabetic or numeric.) Where both data types must appear in the same column, an **n** to select the data type for the entry is preferred; numeric columns center text in column entries, but

alphabetic columns do not align numbers.

The end of the format section is indicated by a period. The layout of the keyletters in the format section resembles the layout of the actual data in the table. Thus a simple format specifying a table of three columns might be:

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

You can give successive lines of a table on consecutive lines of the format section as shown in the preceding example, or you can specify successive line formats on the same line, separated by commas, so that the format for the preceding example might have been written:

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

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. (Notice that the keyletter for each column format is separated by a space for clarity. You can also effectively separate keyletters and keyletter parameters by using tabs.) A sample table in this format might be:

OVERALL TITLE

Item-a		34.22	9.1
Item-b		12.65	.02
Items:	c,d,e	23	5.8
Totals		69.87	14.92

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; **tbl** ignores columns of data beyond that number and does not warn you.

Note: Difficulties with formatting sometimes arise because all table entries are assumed by **tbl** to be in the font and size in use when the **.TS** command was encountered, except those font and size changes indicated in the table format section and in the table data (as in the entry \s+3\fIData\fP\s0). This can cause problems in calculation of column widths. Therefore, although you can insert **troff** requests within a table, you must take special care to avoid confusing **tbl** by using requests that alter column width calculations. Use requests such as **.ps** with care.

If the format of a table must be changed after many similar lines, as with sub-headings or summarizations, you can use the **.T&** (table continue) command to change column parameters. The outline illustrating 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.

Notes:

1. Although you can use any number of lines in a table, only the first 200 lines can be used by **tbl** in setting up the table; you can arrange a multiple-page table as several single-page tables if this proves to be a problem.

2. You cannot change these selections within a table:

   The global options (such as **box**)

   The number of columns

   The space between columns

   The selection of columns to be made equal-width.

   You can, of course, break the table up into several tables to accomplish similar purposes.

There are some additional features of the keyletter system.

Note: The order of the following features is immaterial; they need not be separated by spaces, except as indicated to avoid ambiguities involving point size and font changes.

Creating Horizontal Lines: You can use an _ (underscore) to enter a horizontal line in a place in a column or an **=** (equal sign) to enter a double horizontal line. This lets you construct table ruling. If an adjacent column also contains a horizontal line, or if there are vertical lines adjoining this column, this horizontal line extends to meet the adjacent lines. If any entry data is given for a row containing horizontal lines, it is ignored and **tbl** prints a warning message. **nroff** produces a single horizontal line where a double horizontal line is requested.

Creating Vertical Lines: You can place a vertical bar (**|**) between column keyletters to cause separation of columns of table data by vertical lines. A vertical bar to the left of the first keyletter or to the right of the last one produces a line at the edge of the table. If two vertical bars appear between keyletters, a double vertical line is drawn.

Separating Columns: You can add a number following a keyletter to indicate the amount of separation between that column and the next column. The number normally specifies the separation in ens (one en is about the width of the letter n). (1) If you use the **expand** parameter, 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 (ens). If separation is changed in a table, the largest space requested for a column governs. See the note at the

end of this list, in topic 2.10.2.3.

Vertical Spanning: Normally, vertically spanned (using (^), as described on page 2.10.2.3) items extending over several rows of the table are centered in their vertical range. If a keyletter is followed by **t** or **T**, any corresponding vertically spanned item will begin at the top line of its range (see Staggering Lines, in topic 2.10.2.3).

Selecting Fonts: You can follow a keyletter with a one or two-character string containing a font name or number. The keyletter **f** or **F** indicating a following font selection parameter is optional; the characters **r**, **i**, **b**, and **s** and their capitalized forms represent selectable fonts. For purposes here, all font names are two letters; a one-letter font name must be separated from whatever follows, even a period, by a space or a tab. The single letters **B**, and **i**, for instance, when followed by a space, are synonyms for **fB** and **fI**. Font change requests given with the table entries override these specifications.

Changing Point Size: You can set the point size of a table entry by following a keyletter with the letter **p** or **P** and a number to indicate the point size of the corresponding table entry. The number may be a signed digit, in which case it is taken as an increment or decrement from the current point size. An unsigned digit is interpreted as an absolute point size specification. See the note at the end of this list, in topic 2.10.2.3.

Changing Vertical Spacing: You can select the vertical line spacing used within a multiple-line table entry by following a keyletter of any entry on a row with a **v** or **V** and a number specifying the number of points of line separation. Spacing is changed between the current and following rows. The number may be a signed digit, in which case it is taken as an increment or decrement from the current vertical spacing, rather than an absolute specification in points. This request has no effect unless the corresponding table entry is a text block (see Text Blocks, in topic 2.10.2.4). See also the note at the end of this list, in topic 2.10.2.3.

Setting Column Width: You can set minimum column width by following a keyletter with the letter **w** or **W** and a numeric width value in parentheses. If the width of the largest element in a column is wider than the minimum specification given, the width of the element is used. The actual width used is also used by **tbl** as a default line length for included text blocks (see Text Blocks, in topic 2.10.2.4). Normal **troff** scales can be used to scale the width value; if no scale is specified, the default scale is ens. If the width specification is an integer without a scale indication the parentheses may be omitted (but see the note at the bottom of this list, in topic 2.10.2.3).

Note: If the width specification value is changed in a column, the last one given for a column is used.

Selecting Equal-width Columns: You can create a group of regularly spaced, equal width columns by following keyletters with the letter **e** or **E**. All columns containing a keyletter that is followed by **e** or **E** are made the same width. The column width will not be less than the widest element of any column selected.

Staggering Lines: You can follow a keyletter with the parameter **u** or **U** to indicate that the corresponding entry is to be moved up one-half

line width. This makes it easy, for example, to have a row of differences between two lines set between them in an adjoining column or to have an entry in one column apply to both lines it is set between. (See also Vertical Spanning, in topic 2.10.2.3.)

Note: The **allbox** option does not work with staggered columns.

Zero-width items: You can follow a keyletter with the letter **z** or **Z** to indicate that the corresponding data item is to be ignored in calculating column widths. This is useful, for example, in allowing headings to run across adjacent columns where spanned headings would be inappropriate or to make a column slightly narrower than it would otherwise be set by the width of the widest entry and column separation limits.

Note: Keyletter parameters that use a numeric value must be separated by a space from other parameters using a numeric value, if the given numeric value is not isolated in some other fashion allowed by the parameter (as in Setting Column Width, in topic 2.10.2.3). Otherwise, the formatter may use a single given numeric value for all parameters accepting numeric values; other supplied values may be ignored.

(1) More precisely, an ***en*** is a number of points (1 point = 1/72 inch approximately) equal to half the current type size. The letter ***n*** in most **troff** types is the nominal character width, as compared to an ***m*** (maximum width) or punctuation mark (narrow width). In **nroff**, an en and an em are equal in size; and that is the nominal character width.

2.10.2.4 Table Data

The data for the table are typed after the format. Normally, you type each line in the table as one line of data. You can break very long input lines by using the backslash (****) as the last character of a line. **troff** treats a line ending with the character **** as a continuation of the previous line (and removes the \ from the data). The data for different columns (the table entries) are separated by tabs or by whatever character you have specified using the **tabs** parameter (see topic 2.10.2.2).

There are a few special cases:

troff requests within tables: An input line that you begin with a period which is followed by anything but a number is assumed to be a request to **troff** and is passed through **tbl** unchanged, retaining its relative position in the table. You can, for example, create space within a table by **.sp** requests to **troff** in the data.

Full width horizontal lines: **tbl** treats an input line containing only an _ (underscore) or **=** (equal sign) character as a request for a single or double line, respectively, that extends the full width of the table.

Single column horizontal lines: **tbl** treats an input table entry containing only an _ (underscore) or **=** (equal sign) character as a request for a single or double line, respectively, that extends the full width of the **column**. **tbl** extends such lines to meet horizontal or vertical lines adjoining the column. To obtain these characters explicitly in a column, either precede them by **\&** or follow them by a space before the usual tab or new-line. **nroff** produces a single horizontal line where a double horizontal line is requested.

Short horizontal lines: **tbl** treats an input table entry containing only the string **_** as a request for a single line as wide as the reserved entry width of the column. The line is not extended to meet adjoining lines.

Repeated characters: An input table entry containing only a string of the form **\Rx** where **x** is any character, is replaced by repetitions of the character **x** as wide as the data reserved data entry width in the column.

Vertically spanned items: An input table entry containing only the character string **\(^)** indicates to **tbl** that the table entry immediately above spans downward over this row. It is equivalent to a table format keyletter circumflex ((^)).

Text blocks: In order to include a block of text as a table entry, precede it by **T{** and follow it by **T}**. This way, you can enter something that cannot be conveniently typed, as a simple string between tabs:

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

Note that you must use the T} end delimiter to begin a line; additional columns of data may follow after a tab on the same line.

If you use more than 30 or so text blocks in a table, various limits in the **troff** program are likely to be exceeded, producing diagnostic messages such as **too many string/macro names** or **too many number registers**.

Text blocks are sent to **troff** by **tbl** and processed separately; then the text block is reinserted into the table. If you do not specify line length for a block of text in the block or in the table format, **tbl** sets the value to L¦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. The other parameters (point size, font, and so on) used in setting the block of text are those in effect at the beginning of the table (including the effect of the **.TS** macro) and any table format specifications of size, spacing, and font, using the **p**, **v**, and **f** parameters to the column keyletters. **troff** recognizes parameter settings within the text block itself, but requests in the table data but not within the text block do not affect that block.

2.10.3 Usage, Input, and Output

The **tbl** command is run from the system level of the AIX PS/2 Operating System. You can use it to create tables of various levels of complexity and use various formatting commands to print them. Following are descriptions and examples of some uses:

To create a sample **nroff** or **col** table:

tbl -TX file | nroff -Tlp | col -x

For more complex use, where there may be several input files tha contain equations and memorandum macros as well as tables, a normal command structure is:

tbl -TX **file1 file2** ... | neqn | nroff -mm -Tlp | col -x

which is equivalent to:

mm -t -e -T745 **file1 file2** ...

A complex **troff** table command structure:

tbl **file1 file2** ... | eqn | troff -mm

which is equivalent to:

mmt -t -e **file1 file2** ...

You can use the usual options on the **troff** and **eqn** commands.

The usage for **nroff** is similar to that for **troff**.

Note: Some printers cannot print boxed tables directly. The **TELETYPE** Model 37, however, can.

If a hyphen is given for a file name, **tbl** reads from standard input at that point in the formatting process.

There is a special **-TX** command-line option to **tbl** which produces output that does not have fractional line motions in it. This is useful if your printer cannot perform backspacing or partial line-feed movements. The only other command-line parameter recognized by **tbl** is **-mm**, which is turned into a command to fetch the memorandum macro file. Usually you will find it more convenient to place memorandum macro calls on the typesetter-call portion of the command line, but they are accepted by **tbl** as well.

Use **tbl** before **eqn** when they are used together on the same file. If there are no equations within tables, either order works, but it is usually faster to run **tbl** first, since **eqn** normally produces a larger expansion of the input than **tbl**. However, if there are equations within tables that use the **delim** mechanism in the **eqn**, **tbl** must be first or the output will be scrambled.

You should probably avoid using equations in **n** columns since **tbl** attempts to split numerical format items about a decimal point, and this is not possible with equations. If for some reason you need to do so, however, the **delim(xx)** table option prevents splitting of numerical columns within the delimiters. For example, if the **eqn** delimiters are **!!**, giving **delim**

(!!) causes a numerical column such as **1245 !+- 16!** to be divided after 1245, not after 16.

tbl accepts up to about 35 columns, but the actual number that you can process may be smaller, depending on availability of **troff** number registers.

Note: Avoid number register names used by **tbl**, which include two-digit numbers from 31 to 99 and strings of the form **4x**, **5x**, **#x**, **x+**, **x|**, **(^)x**, and **x-**, where **x** is any lowercase letter. The names **##**, **#-**, and **#(^)** are also used in certain circumstances.

For aid in writing layout macros, **tbl** defines a number register **TW** that is the table width; it is defined by the time that the **.TE** macro is invoked and may be used in the expansion of that macro.

The **macro T#** is defined to assist in laying out multiple-page boxed tables. It produces the bottom lines and side lines of a boxed table and is invoked at its end. You can use this macro in a page footer to box a multiple-page table. In particular, **mm** macros are used to print a multiple-page boxed table with a repeated table heading by giving the argument **H** to the **.TS** macro. If the table start macro is defined as **.TS H**, then after the options, if any, and format, the table heading must be specified with **.TH**, marking the end (then comes the data). Material up to the **.TH** is placed at the top of each page of the table. Note that this is *not* a feature of **tbl**, but of the memorandum macros (see Chapter 7, "Memorandum Macros").

2.10.4 Examples

The following are examples of **tbl**. The &DIAMOND. symbol and surrounding spaces are used in input to represent an input TAB character.

Subtopics
2.10.4.1 Input:
2.10.4.2 Output:
2.10.4.3 Input:
2.10.4.4 Output:
2.10.4.5 Input:
2.10.4.6 Output:
2.10.4.7 Input:
2.10.4.8 Output:
2.10.4.9 Input:
2.10.4.10 Output:
2.10.4.11 Input:
2.10.4.12 Output:
2.10.4.13 Input:
2.10.4.14 Input:
2.10.4.15 Output:

2.10.4.1 Input:

```
.TS
allbox;
C S S
C C C
N N N.
Common Stock
Year&DIAMOND.Price&DIAMOND.Dividend
1981&DIAMOND.41-54&DIAMOND.$2.60
2&DIAMOND.41-54&DIAMOND.2.70
3&DIAMOND.46-55&DIAMOND.2.87
4&DIAMOND.51-59&DIAMOND..95*
.TE

*(first quarter only)
```

2.10.4.2 Output:

```
+-----------------------------------+
¦           Common Stock            ¦
+-----------------------------------¦
¦ Year     ¦ Price    ¦ Dividend    ¦
+----------+----------+-------------¦
¦ 1981     ¦ 41-54    ¦  $2.60      ¦
+----------+----------+-------------¦
¦    2     ¦ 41-54    ¦   2.70      ¦
+----------+----------+-------------¦
¦    3     ¦ 46-55    ¦   2.87      ¦
+----------+----------+-------------¦
¦    4     ¦ 51-59    ¦    .95(*)   ¦
+-----------------------------------+

(*) (first quarter only)
```

2.10.4.3 Input:

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

Bridge&DIAMOND.Designer&DIAMOND.Length

Brooklyn&DIAMOND.J. A. Roebling&DIAMOND.1595
Manhattan&DIAMOND.G. Lindenthal&DIAMOND.1470
Williamsburg&DIAMOND.L. L. Buck&DIAMOND.1600

Queensborough&DIAMOND.Palmer &&DIAMOND.1182
&DIAMOND. Hornbostel

&DIAMOND.&DIAMOND.1380
Triborough&DIAMOND.O. H. Ammann&DIAMOND.
&DIAMOND.&DIAMOND.383

Bronx Whitestone&DIAMOND.O. H. Ammann&DIAMOND.2300
Throgs Neck&DIAMOND.O. H. Ammann&DIAMOND.1800
.TE
```

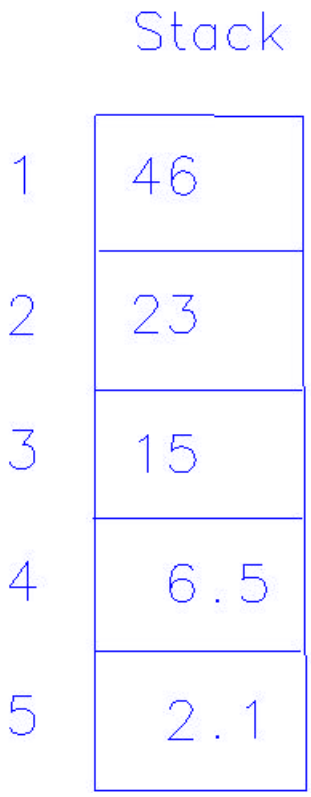
2.10.4.4 Output:

Major New York Bridges		
Bridge	Designer	Length
Brooklyn Manhattan Williamsburg	J. A. Roebling G. Lindenthal L. L. Buck	1595 1470 1600
Queensborough	Palmer & Hornbostel	1182
Triborough	O. H. Ammann	1380 383
Bronx Whitestone Throgs Neck	O. H. Ammann O. H. Ammann	2300 1800

2.10.4.5 Input:

```
.TS
c c
np-2 | n | .
&DIAMOND.Stack
&DIAMOND._
1&DIAMOND.46
&DIAMOND._
2&DIAMOND.23
&DIAMOND._
3&DIAMOND.15
&DIAMOND._
4&DIAMOND.6.5
&DIAMOND._
5&DIAMOND.2.1
&DIAMOND._
.TE
```

2.10.4.6 Output:



2.10.4.7 Input:

```
.TS
box;
L L L
L L _
L L | LfB
L L _
L L L.
January&DIAMOND.February&DIAMOND.March
April&DIAMOND.May
June&DIAMOND.July&DIAMOND.MONTHS
August&DIAMOND.September
October&DIAMOND.November&DIAMOND.December
.TE
```

2.10.4.8 Output:

2.10.4.9 Input:

```
.TS
allbox;
CfI S S
C CW(2i) CW(2i)
L L L.
New York Area Rocks
.sp
Era&DIAMOND.Formation&DIAMOND.Age (years)
Precambrian&DIAMOND.Reading Prong&DIAMOND.>1 billion
Paleozoic&DIAMOND.Manhattan Prong&DIAMOND.400 million
Mesozoic&DIAMOND.T{
.na
Newark Basin, incl.
Stockton, Lockatong, and Brunswick
formations
.ad
T}&DIAMOND.200 million
Cenozoic&DIAMOND.Coastal Plain&DIAMOND.T{
.na
On Long Island 30,000 years;
Cretaceous sediments redeposited
by recent glaciation
.ad
T}
.TE
```

2.10.4.10 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

2.10.4.11 Input:

```
.TS
box;
cfB s s s.
Composition of Foods
_
.T&
c | c s s
c | c s s
c | c|c | c.
Food&DIAMOND.Percent by Weight
\^&DIAMOND._
\^&DIAMOND.Protein&DIAMOND.Fat&DIAMOND.Carbo-
\^&DIAMOND.\^&DIAMOND.\^&DIAMOND.hydrate
_
.T&
L | N | N | N.
Apples&DIAMOND..4&DIAMOND..5&DIAMOND.13.0
Halibut&DIAMOND.18.4&DIAMOND.5.2&DIAMOND....
Lima beans&DIAMOND.7.5&DIAMOND..8&DIAMOND.22.0
Milk&DIAMOND.3.3&DIAMOND.4.0&DIAMOND.5.0
Mushrooms&DIAMOND.3.5&DIAMOND..4&DIAMOND.6.0
Rye bread&DIAMOND.9.0&DIAMOND..6&DIAMOND.52.7
.TE
```

2.10.4.12 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

2.10.4.13 Input:

```
.TS
C S
Cip-2 S
L N
A N.
Some London Transport Statistics
(Year 1964)
Railway route miles&DIAMOND.244
Tube&DIAMOND.66
Sub-surface&DIAMOND.22
Surface&DIAMOND.156
.sp .5
.T&
L r
A R.
Passenger traffic \- railway
Journeys&DIAMOND.674 million
Average length&DIAMOND.4.55 miles
Passenger miles&DIAMOND.3,066 million
.T&
l r
a r.
Passenger traffic \- road
Journeys&DIAMOND.2,252 miles
Average length&DIAMOND.2.26 miles
Passenger miles&DIAMOND.5,094 million
.T&
L N
A N.
.sp .5

Vehicles&DIAMOND.12,521
Railway motor cars&DIAMOND.2,905
Railway trailer cars&DIAMOND.1,269
Total railway&DIAMOND.4,174
Omnibuses&DIAMOND.8,347
.T&
L N
A N.
.sp .5
Staff&DIAMOND.73,739
Administrative, etc.&DIAMOND.5,582
Civil engineering&DIAMOND.5,134
Electrical eng. \- railway&DIAMOND.4,310
Mech. eng. \- road&DIAMOND.9,152
Railway operations&DIAMOND.8,930
Road operations&DIAMOND.35,946
Other&DIAMOND.2,971
.TE
```

Some London Transport Statistics
(Year 1964)

Railway route miles	**244**
Journeys	**66**
Sub-surface	**22**

Surface	**156**
Passenger traffic -- railway	
Journeys	**674** million
Average length	**4.55** miles
Passenger miles	**3,066** million
Passenger traffic -- road	
Journeys	**2,252** million
Average length	**2.26** miles
Passenger miles	**5,094** million
Vehicles	**12,521**
Railway motor cars	**2,905**
Railway trailer cars	**1,269**
Total railway	**4,174**
Omnibuses	**8,347**
Staff	**73,739**
Administrative, etc.	**5,582**
Civil engineering	**5,134**
Electrical eng.	**1,714**
Mech. eng. -- railway	**4,310**
Mech. eng. -- road	**9,152**
Railway operations	**8,930**
Road operations	**35,946**
Other	**2,971**

2.10.4.14 Input:

```
.TS
box, tab(:);
cb s s s s
cP-2 s s s s
c || c | c | c | c
c || c | c | c | c
r2 || n2 | n2 | n2 | n.
Readability of Text
Line Width and Leading for 10-Point Type
=
Line:Set:1-Point:2-Point:4-Point
Width:Solid:Leading:Leading:Leading:
_
9 Pica:\-9.3:\-6.0:\-5.3:\-7.1
14 Pica:\-4.5:\-0.6:\-0.3:\-1.7
19 Pica:\-5.0:\-5.1:\0.0\-2.0
31 Pica:\-3.7:\-3.8:\-2.4:\-3.6
43 Pica:\-9.1:\-9.0:\-5.9:\-8.8
.TE
```

2.10.4.15 Output:

Readability of Text

Line Width and Leading for 10-Point Type

Line Width	Set Solid	1-Point Leading	2-Point Leading	4-Point Leading
9 Pica	-9.3	-6.0	-5.3	-7.1
14 Pica	-4.5	-0.6	-0.3	-1.7
19 Pica	-5.0	-5.1	0.0	-2.0
31 Pica	-3.7	-3.8	-2.4	-3.6
43 Pica	-9.1	-9.0	-5.9	-8.8

2.10.5 Keywords

The following table describes the **tbl** keywords:

Keyword	Effect of the Keyword
a or **A**	Sets alphabetic subcolumn entry (see topic 2.10.2.3).
allbox	Boxes each entry (see topic 2.10.2.2).
b or **B**	Boldfaces entries (see topic 2.10.2.3).
box	Encloses entire table in a box (see topic 2.10.2.2).
c or **C**	Centers the entry in the column (see topic 2.10.2.3).
center	Centers table horizontally in (offset and margined) page (see topic 2.10.2.2).
delim (xy)	Defines **eqn** delimiters.
doublebox	Encloses the table in a double box (see topic 2.10.2.2).
e or **E**	Makes columns equal width (see topic 2.10.2.3).
expand	Makes the table the offset and margined line width (see topic 2.10.2.2).
f or **F**	Sets the font type.
i or **I**	Makes the item italic font.
l or **L**	Aligns column left (see topic 2.10.2.3).
linesize (num)	Sets the line length (see topic 2.10.2.2).
n or **N**	Makes the column a numeric column (see topic 2.10.2.3).
num	Specifies column separation (see topic 2.10.2.3).
p or **P**	Sets type size (see topic 2.10.2.3).
r or **R**	Aligns column right.
s or **S**	Spans entry across columns (see topic 2.10.2.3).
t or **T**	Places vertically spanned item at top of space (see topic 2.10.2.3).
tab (char)	Sets the data separation character to **char**.
T{ ... T}	Marks a text block.
u or **U**	Staggers column entry relative to previous row (see topic 2.10.2.3).
v or **V**	Sets the vertical spacing (see topic 2.10.2.3).
w or **W**	Assigns a minimum column width value (see topic 2.10.2.3).
z or **Z**	Specifies a Zero width item (see topic 2.10.2.3).
.xx	Calls a formatter request or another macro (if not the **tbl** command) of the name **xx**.
|	Creates a vertical line (see topic 2.10.2.3).
||	Creates a double vertical line (see topic 2.10.2.3).
(^)	Spans an entry vertically to previous row (see topic 2.10.2.3).
\(^)	Spans vertically (see topic 2.10.2.4).
=	Creates a double horizontal line (see topic 2.10.2.3).
_	Creates a horizontal line (see topic 2.10.2.4).
_	Creates an underscore line the width of an en (see topic 2.10.2.4).
\Rx	Repeats character **x**.
-	Uses standard input at this location.

2.11 Chapter 11. Formatting Viewgraphs and Slides

Subtopics

2.11.1 About This Chapter

This chapter describes a package of **troff** formatter macros called viewgraph macros that are designed for typesetting for the production of viewgraphs and slides. The sheets of printed output used to prepare the viewgraphs and slides (and sometimes the viewgraphs and slides themselves) are called ***foils***.

When a foil is printed, the actual projection area is marked at the corners by ***cross hairs*** (plus signs) that correspond to the inside corners of a viewgraph mount. This is an aid in positioning the foil for mounting.

All foils other than the **.VS** (square viewgraph) foil also have a set of horizontal and vertical ***crop mark*** lines. These indicate how much of the foil will be seen if it is made into a 35 mm slide, rather than into a viewgraph.

You can prepare transparent plastic ***viewgraphs*** in a variety of dimensions, as well as 35 mm and 2 ¦ 2 inch slides, using the viewgraph macros. You can control styles, fonts, title sizes, highlighting, and subordination levels. Because the foils are typeset from a text file, you can change the file to include new data or incorporate data into a new presentation. You can perform normal text processing functions by passing text through **spell**, **eqn**, **tbl**, **cw**, or other text processing or preprocessing commands.

Note: You cannot include artwork, graphics, or multicolored text in foils made with this macro package except by manual cut-and-paste methods.

You cannot use the viewgraph macros for Japanese text as these macros do not support multibyte characters.

2.11.2 Making Actual Viewgraphs and Slides

Typesetter output is printed on ***mechanical paper***. On a typical phototypesetter, the paper is a white, opaque photographic paper with black lettering. There are a number of processes for making transparent viewgraphs from opaque paper.

Note: Mechanical paper is heat sensitive. Because some transfer processes involve heat, you should first make copies of the typesetter output and then use these copies for making transparencies.

Slides can be made by photographing foil output. You can have both positive (opaque letters on transparent background) and negative (transparent letters on opaque background) slides made, as well as highlighted and colored-background slides.

There is equipment that lets you create slides from video images. This does not necessarily require output created using viewgraph macros, but you can use them (see "Workstation Screen Output" in topic 2.11.3.2).

2.11.3 Viewgraph Macros

You can use the viewgraph macro package for phototypeset output or obtain an approximation of typeset output to a workstation screen.

Subtopics
2.11.3.1 Typeset Output
2.11.3.2 Workstation Screen Output

2.11.3.1 Typeset Output

You can use the **mvt** command to use the viewgraph macro package with the **troff** typesetter for phototypeset output by entering:

```
mvt [ flags ] file ...
```

The **file** contains text and macro invocations for the foils and **flags** can be one or more of the following:

-a Displays output on a terminal (other than a Tektronix 4014--see "Workstation Screen Output" in topic 2.11.3.2). This is an approximation of typeset output using ASCII characters and symbols.

-e Calls **eqn**.

-t Calls **tbl**.

A hyphen (-) given in place of a **file** causes the **mvt** command to read the standard input (rather than a file), as in the following entry calling the **cw** preprocessor (see "Constant-Width Program Examples" in topic 2.11.4.3):

```
cw [ flags ] file ... | mvt [ flags ] -
```

2.11.3.2 Workstation Screen Output

You can obtain an approximation of typeset output to a workstation screen by using the **mvt** command:

```
mvt -a file ...
```

The video display shows the formatted foils except that:

- Point-size changes are not visible
- Different fonts are not visible
- Titles that are too long can appear proper
- All horizontal motions are reduced to one horizontal space to th right.
- All vertical motions are reduced to one vertical space down

For example, lines of text following a **.B**, **.C**, or **.D** macro do not appear to align properly; although they do align correctly when printed. Although alignment is not correct in this approximation, line breaks and the amount of vertical space used by the text can be observed.

If the foil is not full, **mvt** prints the number of blank lines (in the current point size) that remain on the foil; if the foil is full, a warning is printed to standard output. If the text did overflow the foil, text is printed below the bottom cross hairs (crop marks) and below the bottom line for slide markings.

2.11.4 The troff Preprocessors

You can use the various **troff** formatter preprocessors to typeset foils when you require more powerful formatting capabilities.

Subtopics
2.11.4.1 Tables
2.11.4.2 Mathematical Expressions
2.11.4.3 Constant-Width Program Examples

2.11.4.1 Tables

You can use the **tbl** program to set up columns of data within a viewgraph or slide. The **.TS** and **.TE** macros are not defined in the viewgraph macro package but are merely flags to call **tbl**.

2.11.4.2 Mathematical Expressions

You can use the **eqn** program to typeset mathematical expressions and formulas on foils if you take care to specify proper fonts and point sizes. The **.EQ** and **.EN** macros are not defined in the viewgraph macro package and call the **eqn** program. The example in topic 2.11.7.7 illustrates using the **eqn** program with viewgraph macros.

2.11.4.3 Constant-Width Program Examples

The constant width font simulates workstation and line printer output and can be effective for some purposes, as in presenting computer topics. The **cw** program, as well as the example in topic 2.11.7.7 illustrates the **cw** preprocessor.

2.11.5 Macros of the Viewgraph Macro Package

The following macros are available within the viewgraph macro package.

Subtopics

2.11.5.1 Default Fonts

The viewgraph macros assume that the Helvetica Regular (also known as Geneva) font is the default font and that it is loaded in physical device position 1 of the typesetter. You can use additional fonts, as well as change the default font using the **.DF** (default font) macro:

```
.DF num font [ num font ... ]
```

The **.DF** macro sets **troff** values appropriate for **font** in position **num**. Because the C/A/T typesetter can have four fonts mounted, you can specify up to four **num font** parameter pairs with **num** being an unrepeated digit from 1 to 4.

Note: The font named first is made the default font. (The viewgraph macro package does not assume font position 1 as the default when you use the **.DF** macro.)

The actual fonts mounted in the typesetter must be installed by the typesetter operator. You can use any available font, but generally a Roman font presents a poorer appearance than Helvetica font for foils.

Note: Not all fonts have the same character proportions. Since the viewgraph macro package is designed for use of Helvetica font, using other fonts can sometimes result in an appearance of irregular character spacing. (See also "Miscellaneous Viewgraph Macros" in topic 2.11.5.10.)

The **.DF** macro must immediately precede a foil-start macro (see "Foil-Start Macros" in topic 2.11.5.2); the initial setting is equivalent to:

```
.DF 1 H 2 I 3 B 4 S
```

The example in topic 2.11.7.7 illustrates the use of the **.DF** macro.

2.11.5.2 Foil-Start Macros

Each foil is started 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 11-1.

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). The 35 mm slides are narrower 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, you can use slide files for viewgraphs, but not (usually) vice versa. On the other hand, viewgraphs can hold a bit more text than slides. Square slides and viewgraphs are the exception, being the same. For that reason, there is no **.SS** (square slide) macro; the **.VS** (square viewgraph) macro serves for square viewgraphs and 2 ¦ 2 slides.

Note: The **.VW** and **.SW** macros produce foils that are 7 ¦ 5.4 inches because commonly available typesetter paper is less than 9 inches wide. These foils must be enlarged by a factor of 9/7 before they can be used as 9 ¦ 7-inch high viewgraphs.

Table 11-1. Viewgraph Start-Macros and Foil Sizes

Macro	Foil Size and Type	Printed Image	Text Area
.VS	7¦7 viewgraph or 2¦2 slide	7¦7	6.8¦6
.Vh	7¦5 portrait viewgraph	7¦5	6.8¦4.2
.Vw	5¦7 landscape viewgraph	5¦7	4.8¦6
.VH	9¦7 portrait viewgraph	9¦7	8.8¦6
.VW	7¦9 landscape viewgraph	5.4¦7	5.2¦6
.Sh	portrait 35mm slide	7¦4.6	6.8¦3.8
.Sw	landscape 35mm slide	4.6¦7	4.4¦6
.SH	portrait 35mm slide	9¦6	8.8¦5
.SW	landscape 35mm slide	4.6¦7	4.4¦6

The printed image in the above table is listed in inches. The size ratios are listed as height ¦ width. The dimensions listed for the slides refer to the size of the machine paper output from which the slides are taken. The 35mm slides in the 7 ¦ 5 and 5 ¦ 7 ratios have proportionately smaller frame apertures than their nominally larger counterparts though the slide external dimensions are the same.

Note: The terms ***portrait*** and ***landscape***, used in reference to image shape, refers to the orientation of a rectangular shape. A portrait image is higher than it is wide, and a landscape image is wider than it is high.

Note: Each foil created by **.VW**, and each foil created by **.SW** if used as a viewgraph, must be enlarged by a factor of 9/7.

Each foil start macro causes the previous foil (if any) to end, 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 dat

<<Company Name>

FOIL **num**.

where **num** is the sequence number in the current ***run***, or printed series, of foils.

Default heading information can be changed by specifying three optional parameters to a foil-start macro:

.XX [**num**] [**ident**] [**date**]

where:

XX stands for one of the nine foil-start macros.

The **num** is the foil identifier (typically a number).

The **ident** is other identifying information (such as the initials of the person creating the foil).

The **date** is usually given as the date.

The resulting heading information consists of three lines of right-justified text:

ident

date

FOIL **num**.

If **date** and **ident** are omitted on a foil-start macro, the corresponding values (if any) from the previous foil-start macro are used.

2.11.5.3 Titles

The **.T** (title) macro creates a centered title from its string parameter:

```
.T "string"
```

You must enclose **string** within double quotes ("...") if it contains spaces. The size of the title is four points larger than the prevailing point size. Any indentation established by the **.I** macro (see "Global Indents" in topic 2.11.5.5) has no effect on titles, which are always centered within the foil horizontal dimension.

Pages 2.11.7.3 and 2.11.7.5 offer examples of the **.T** macro.

2.11.5.4 Level Macros

There are four viewgraph macros to provide indentation levels:

.A

.B

.C

.D

Each level macro places the text that follows it at the corresponding level of indentation (level A for **.A**, for example). Each level macro also sets the amount of vertical spacing. The amount of vertical spacing done by each level macro is controlled by the **.DV** (vertical distance) macro (see "Default Vertical Space" in topic 2.11.5.7).

The example in topic 2.11.7.5 illustrates the use of the level macros.

Subtopics
2.11.5.4.1 The A Level
2.11.5.4.2 The B Level
2.11.5.4.3 The C Level
2.11.5.4.4 The D Level

2.11.5.4.1 The A Level

The **.A** macro sets the left margin.

```
.A [x]
```

Each **.A** macro spaces a half line from the preceding text, unless the **x** parameter 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 (or A level) macro. The A level is automatically called by each of the foil start macros. Repeated **.A** calls are ignored, but each successive call of any of the other three level macros generates the corresponding mark and indentation.

You can also call the **.A** macro through the **.I** macro (see "Global Indents" in topic 2.11.5.5).

2.11.5.4.2 The B Level

The B level heading mark and indentation is set by the **.B** (B level) macro:

```
.B ["mark" [size]]
```

The B level items are marked by a bullet printed in slightly reduced point size by default. The text that follows the **.B** is spaced a half line below the preceding text.

You can change the B level **mark** by specifying the desired character string to be used as the **mark**. You must enclose all string parameters that contain spaces within double quotes, but the quote marks are optional if no spaces occur in the string. The following example produces a numbered list:

```
.VS
This is a list of items:
.B 1.
This is item number 1.
.B 2.
This is item number 2.
.B 3.
This is the third and last item on this foil.
```

You can change the point size of the **mark** with the **size** parameter. The **size**, if given, is a numeric value that alters the desired type size. If you give a **.B** macro without the **size**, the current type size is used. An unsigned or positive parameter increases type size, and a negative parameter decreases type size. The type size is changed relative to the current type size in points. A **size** parameter greater than **99** causes **mark** to be reduced in size as if it were the default (bullet) **mark**. After the **mark** is printed, the previous point size is restored. These point-size changes occur automatically.

2.11.5.4.3 The C Level

The C level is like the B level except that it is indented farther to the right and the default mark is an -- em dash (as called by **\(em)** in a slightly reduced point size. **.C** is the C level macro.

2.11.5.4.4 The D Level

The **.D** (D level) macro 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 by causing a break. It formats like the B and C level macros otherwise. The D level default mark is a bullet smaller than that used for the B level.

2.11.5.5 Global Indents

You can shift the entire text of the foil (except titles) right or left using the **.I** (indent) macro.

```
.I [indent] [a [num]]
```

The **indent** parameter can be a signed positive or negative numeric value, indicating right or left movement from the current margin. If unsigned, the parameter specifies the new margin, relative to the initial default margin. If the **indent** value is not dimensioned, it is assumed to be in inches. If the **indent** is null or omitted, **0i** is the assumed value, causing the margin to revert to the initial default margin.

If you specify the **a** parameter, the **.I** macro calls the **.A** macro (see "The A Level" in topic 2.11.5.4.1) before exiting. The third parameter value, **num**, if present, is a parameter to the **.A** macro.

The examples in topics 2.11.7.3 and 2.11.7.7 illustrate uses of the **.I** macro.

2.11.5.6 Point Sizes and Line Lengths

Each foil-start macro has an appropriate default type point size and line length for its overall dimensions. You can set point size and line length by calling the **.S** (size) macro:

.S [**tsize**] [**llength**]

Do not give a third parameter to **.S**. You can give a type size in points as the first parameter. If **tsize** is:

A null parameter, the previous point size is restored
A signed negative number, the point size is reduced by the specifie amount.
A signed positive number, point size is increased by the specifie amount.
An unsigned number, it is used as the new point size
A number greater than 99, the initial default point size is restored

Vertical spacing is always set at 1.25 times the current point size for viewgraphs and slides.

If you give a second parameter, it specifies line length.

The value can be dimensioned
If it is not dimensioned and is less than 10, it is taken as inches
If it is not dimensioned and is greater than or equal to 10, it i taken as **troff** machine units (1/432 of an inch, horizontally).

The **.S** macro changes point size and vertical spacing immediately, but a line-length change requested with this macro does not take effect until the next level macro call.

The example in topic 2.11.7.7 illustrates the **.S** macro.

2.11.5.7 Default Vertical Space

The ***default vertical space macro***, **.DV**, lets you change the vertical spacing done by each of the four level macros (see "Level Macros" in topic 2.11.5.4):

```
.DV [aval] [bval] [cval] [dval]
```

The first parameter is the spacing value for the **.A** macro, the second is for the **.B** macro, and so on. Null parameters leave the corresponding spacing unaffected. All given parameters must be dimensioned. The initial setting is equivalent to:

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

2.11.5.8 Underlining

You can underline using the **.U** (underline) macro:

```
.U string1 [string2]
```

The **.U** macro takes one or two parameters. The first parameter is the string of characters to be underlined. If the underlined string contains a blank space, you must enclose it in quotes (for example, ". . .."). The second parameter, if present, is not underlined but is appended to the first parameter. For example:

```
.U phototypesetter
```

produces:

phototypesetter

while:

```
.U under line
```

produces:

underline

2.11.5.9 Text Filling, Adjusting, and Hyphenation

By default, the viewgraph macros fill, but neither adjust nor hyphenate text. This seems to work well for most foils. You can change the defaults by using the **.AD**, **.FI**, **.HY**, **.NA**, **.NF**, and **.NH** macros that call their lowercase formatter equivalents (see "troff Requests and Viewgraph Macro Synonyms" in topic 2.11.5.11).

2.11.5.10 Miscellaneous Viewgraph Macros

The ~ (tilde) is defined by the viewgraph macros as a ***constant space***; you can use the tilde wherever you wish a fixed size, nonadjustable space in your text. To override this definition, you can include the following formatter (translate) request in the input file (see topic 2.8.16.3):

```
.tr ~ ~
```

The formatter string call ***(Tm** works within **mvt** and generates a trademark symbol or string.

2.11.5.11 troff Requests and Viewgraph Macro Synonyms

In general, you should not intermix **troff** requests with the viewgraph macros because this often leads to undesirable (and sometimes astonishing) results. You should generally use the uppercase synonyms that have been defined in the viewgraph macro package. You should use other **troff** formatter requests sparingly if at all; you will find that some cannot be used or only work in certain cases. The requests that affect point size, indentation, page offset, line and title lengths, and vertical spacing between lines, are likely to cause problems. You should use the **.S** and **.I** viewgraph macros instead.

The viewgraph macro package recognizes the following uppercase viewgraph macros as the synonyms of the corresponding lowercase **troff** requests.

Table 11-2. Viewgraph Macros that Convert to Typesetter Requests

Macro	**Label**	**Meaning and Reference**
.AD	Adjust	Begins line adjustment (text justification) when **fill** is on (see "Text Filling, Adjusting, and Hyphenation" in topic 2.11.5.9).
.BR	Break	Stops filling the line currently being collected and prints it without adjustment. Following text appears on a new line (see **.br** in topic 2.8.15.4).
.CE	Center	Centers the following line or lines (see **.ce** in topic 2.8.15.4).
.FI	Fill	Begins filling output lines (see **.fi** in topic 2.8.15.4, and "Text Filling, Adjusting, and Hyphenation" in topic 2.11.5.9).
.HY	Hyphenate	Begins automatic hyphenation (see "Text Filling, Adjusting, and Hyphenation" in topic 2.11.5.9).
.NA	No Adjust	Turns adjust mode (right margin justification) off, leaving right margin ragged (see **.na** in topic 2.8.5.2, and "Text Filling, Adjusting, and Hyphenation" in topic 2.11.5.9).
.NF	No Fill	Turns fill mode off (see **.nf** on 2.8.15.4, and "Text Filling, Adjusting, and Hyphenation" in topic 2.11.5.9).
.NH	No Hyphen	Turns automatic hyphenation off (see "Text Filling, Adjusting, and Hyphenation" in topic 2.11.5.9).
.NX	Next File	Stops processing the current file and goes to the next file (see **.nx** in topic 2.8.18.1).
.SO	Switch Source	Begins processing a new file and returns to the current file when finished (see **.so** in topic 2.8.18.1).

.SP	Space Vertically	Spaces vertically in the direction indicated in a parameter (see **.sp** in topic 2.8.15.4).
.TA	Tab Set	Sets tab stops and types (see **.ta** in topic 2.8.16.2.2).
.TI	Temporary Indent	Indents the output text line following the next newline in relation to the current indent level (see **.ti** in topic 2.8.15.4).

2.11.5.12 Viewgraph Macros That Cause 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 parameter. All other viewgraph macros always cause a break. The **troff** formatter synonyms (see "troff Requests and Viewgraph Macro Synonyms" in topic 2.11.5.11) **.AD**, **.BR**, **.CE**, **.FI**, **.NA**, **.NF**, **.SP**, and **.TI** also cause a break, and other **troff** requests may cause a break.

2.11.5.13 Reserved Macro Names

Certain names are used internally by this macro package. In particular, all two-character names starting with either **)** or **]** are reserved. You cannot define your own macros that have the same names as the viewgraph macros and string registers, or any macros that have the same names as **troff** requests, string registers, or macros except the user-defined macros. If you use any of the preprocessors (see "The troff Preprocessors" in topic 2.11.4) or other macro packages, you must also avoid their reserved names. (For some advice on reserved names, see "User-Definable Names" in topic 2.7.11.3.)

2.11.6 Composing Foils

The following are tips for composing foils.

The most useful foils are created using **.VS**, **.Vw**, and **.Sw**, because most projection screens are either square or landscape style (wider than they are tall). The resulting foils require no enlargement before use.

Do not reduce point size below the default value. The default poin size for each type of foil is by design the smallest point size that will result in a foil that is legible by an audience of more than a dozen people in a typical small meeting room. If necessary, use two or more foils rather than reducing the point size. Less text per foil often results in a more effective presentation.

Avoid numerous font changes. A foil with more than two typeface tends to distract the viewer.

Avoid underlined text. Even though the viewgraph macro packag contains a macro for underlining, you should use it very rarely. Highlighting with a different font usually looks better than underlining.

Helvetica sans-serif font is thicker and easier to read for viewgrap presentations than the Roman serif font normally used for typesetting. Times Roman font does let you squeeze more text onto a foil, but visual presentations are usually more effective with sparse text. If you wish to use italic and/or bold typefaces, either the Helvetica regular, italic, and medium fonts, set up by:

```
.DF 1 H 2 HI 3 HM
```

or the Times Roman regular, italic, and bold set up by:

```
.DF 1 R 2 I 3 B
```

are effective. Helvetica Medium is a fairly heavy typeface. True bold typefaces can appear very heavy in visual presentations. The following table identifies fonts used in the example in topic 2.11.7.1 and 2.11.7.3 respectively.

1, 2, and 3 H (default)
4, 7 R and I
5, 6 R and CW

You can use the **.SP** macro to get a greater top margin (for instance, **.5v** or **1v** vertical space) for the foil.

Use both uppercase and lowercase text, which is more readable tha uppercase text only (unless the foil is set in a point size so small that lowercase characters are illegible). More information will fit on the foil without crowding.

Foil style for a presentation should be as consistent as possible Changing fonts, point sizes, foil shapes, and so on, from foil to foil tends to distract the viewer. You should reserve stylistic changes for deliberate emphasis.

To summarize, when in doubt:

```
Do not change point sizes
Do not change fonts or typefaces
Do not underline
Use many sparse foils rather than a few crowded 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 tex
only.
```

2.11.7 Examples

The foil formatting process is illustrated with the following examples.

Subtopics
2.11.7.1 Input
2.11.7.2 Output
2.11.7.3 Input
2.11.7.4 Output
2.11.7.5 Input
2.11.7.6 Output
2.11.7.7 Input
2.11.7.8 Output
2.11.7.9 Input
2.11.7.10 Output

2.11.7.1 Input

```
.Sw
Six stages of a project:
.B
wild enthusiasm
.B
disillusionment
.B
total confusion
.B
search for the guilty
.B
punishment of the innocent
.B
promotion of the non-participants
```

If the preceding text file is given the file name **trivial**, the following command generates the next output:

```
mvt trivial
```

2.11.7.2 Output

5/21/82
BTL
FOIL 1

Six stages of a project:

wild enthusiasm

disillusionment

total confusion

search for the guilty

punishment of the innocent

promotion of the non-participants

2.11.7.3 Input

```
.Vw 2 " Less Trivial " " June 29, 1980 "
.T " What the Walrus Said "
"The time has come," the Walrus said,
.BR
"To talk of many things:
.I .5
.B
Of shoes\(emand ships\(emand sealing wax\(em
.B
Of cabbages\(emand kings\(em
.B
And why the sea is boiling hot\(em
.B
And whether pigs have wings."
```

2.11.7.4 Output

June 29,1980
Less Trivial
FOIL 2

+ +
- -

What the Walrus Said

"The time has come," the Walrus said,
"To talk of many things:

Of shoes--and ships--and sealing
wax--

Of cabbages--and kings--

And why the sea is boiling hot--

And whether pigs have wings."

- -
+ +

2.11.7.5 Input

```
.Vh 3 " Levels & Marks "
.T " Foil Levels & Level Marks "
This is the .A (left margin) level;
.B
This is the .B level,
.B
as is this;
.C
this is the .C level,
.C
as is this;
.D
and this is the .D level,
.D
as is this.
.A
The large bullet, the dash, and the
small bullet are the default "marks" for
levels .B, .C, and .D, respectively.
However, these three levels can also
be marked arbitrarily:
.B B.
Like this (this is the (.B level);
.C 3.
like this (this is the .C level);
.D d.
as is this (this is the .D level).
.B
An arbitrary number of lines of text
can be included in any item at any level;
the text will be filled, but neither adjusted
nor hyphenated, just like this .A level item.
```

2.11.7.6 Output

June 29, 1980
Levels & Marks
FOIL 3

```
 |                                                                  |
+                                                                    +
       Foil Levels & Level Marks

       This is the .A (left margin) level;

          this is the .B level,

          as is this;
          --  this is the .C level,

          --  as is this;
              (.)  and this is the .D level,
              (.)  as is this.

       The large bullet, the dash, and the small
       bullet are the default "marks" for levels .B,
       .C, and .D, respectively.  However, these
       three levels can also be marked arbitrarily:

       B.  Like this (this is the .B level);

           3.  like this (this is the .C level);
              d.  as is this (this is the .D level).

       An arbitrary number of lines of text can
       be included in any item at any level;
       the text will be filled, but neither
       adjusted not hyphenated, just like this
       .A level item.
+                                                                    +
 |                                                                  |
```

2.11.7.7 Input

In the following input the symbol &DIAMOND. represents the tab character:

```
  .de CW
  .I.5 a
  .NF
  ..
  .de CN
  .FI
  .I 0 a
  ..
  .DF 1 R 2 I 3 CW
  .VS 5 " CW & EQN "
  .EQ
  gsize 18
  .EN
  .S 100 5.5
  Input:
  .CW
  .EQ
  sum from k=1 to inf m sup k-1
  ~=~ 1 over 1-m
  .EN
  .CN
  Output:
  .I 2 a
  .EQ
  sum from k=1 to inf m sup k-1
  ~=~ 1 over 1-m
  .EN
  .I 0 a
  Input:
  .CW
  The equation $ f(t) ~=~ 2 pi
  int sin ( omega t ) dt $
  is used here in running text,
  rather than being displayed.
  .CN
  Output:
  .I .5 a
  .EQ
  delim $$
  .EN
  .AD
  The equation $ f(t) ~=~ 2 pi
  int sin ( omega t ) dt $
  is used here in running text,
  rather than being displayed.
  .EQ
  delim off
  gsize 10
  .EN
```

2.11.7.8 Output

June 29, 1980
CW & EQN
FOIL 5

+ +

Input:

```
.EQ
sum from k=1 to inf m sup k-1
~=~ 1 over 1-m
.EN
```

Output:

$$\sum_{k=1}^{\infty} m^{k-1} = \frac{1}{1-m}$$

Input:

```
The equation  $ f(t) ~=~ 2 pi
int sin (omega t ) dt $
is used here in running text,
rather than being displayed.
```

Output:

The equation $f(t) = 2\pi \int \sin(wt)\,dt$ is used here in running text, rather than being displayed.

+ +

2.11.7.9 Input

In the following input the symbol &DIAMOND. represents the tab character:

```
.VS 7 " The Works: Output "
.EQ
delim $$
gsize 14
.EN
Output:
.I 0 a
.SP
.TS
center doublebox;
Cip+4 | Cip+4 S S
^ | L L L
^ | C | C | C
^ | C | C | C
Li | C | C | N.
Users&DIAMOND.Hardware
&DIAMOND._&DIAMOND._&DIAMOND._
&DIAMOND.Computer&DIAMOND.Model&DIAMOND.Serial
&DIAMOND.System&DIAMOND.\^&DIAMOND.Number
=
OS Dev.&DIAMOND.A&DIAMOND.1234&DIAMOND.54
SGS Dev.&DIAMOND.B&DIAMOND.4427A&DIAMOND.3275
Low-End&DIAMOND.C&DIAMOND.XYZ&DIAMOND.221
_
And now ...&DIAMOND.T{
.NA
Some filled text and an equation:
T}&DIAMOND.T{
$ zeta (s) = prod
from k=1 to inf k sup -s $
.AD
T}&DIAMOND.1.2
.TE
```

2.11.7.10 Output

June 29, 1980
The Works: Output
FOIL 7

Output:

Users	*Hardware*		
	Computer System	Model	Serial Number
OS Dev. *SGS Dev.* *Low-End*	A B C	1234 427A XYZ	54 3275 221
And now ...	Some filled text and an equation:	$\zeta(s)=\prod_{k=1}^{\infty} k^{-s}$	1.2

3.0 Part 3. Text Formatting Reference

Subtopics

3.12 Chapter 12. Useful Text Processing Commands

Subtopics

3.12.1 About This Chapter

The following are AIX PS/2 Operating System commands that you may find useful for text processing and formatting. They are grouped by type of activity. Details of command usage are given in the *AIX Operating System Commands Reference*. Useful commands for formatting and preparing C language programs are given in the *AIX PS/2 Operating System Programming Tools and Interfaces*. The Source Code Control System (SCCS) therein documented can be used for some purposes for text formatting, particularly for document revision control.

3.12.2 Copying Files

Command Line	Description
cat file [file] ... > outfile	Reads **file** ... into **outfile**. **Note:** Pre-existing data in **outfile** is lost. If, for example, you give the command **cat file1 > file2**, **file2** is lost.
copy infile outfile **OR** **copy infile [infile] ... directory**	Copies files.
cp	Same as **copy** command.
dd [file-specs] [blocksizes] [conv=parm]	Converts and copies a file.
ln file newfile **OR** **ln files ... directory**	Links files to files or directories; links subdirectories of common directory.
mv files directory **OR** **mv file-or-dir newname**	Moves or renames files or directories.

3.12.3 Creating, Editing, and Deleting Files

Command Line	Description
awk [parms] [file] ...	Finds lines in files matching specified patterns and performs specified actions on them.
cat - > file	Redirects standard input to **file**. End standard input with **Ctrl-D**.
del [-] file ...	Deletes **file** (confirmation required, but can also delete a write-protected file). (Use the **rmdir** command to delete a directory.)
e [file [flags]]	Edits text using a full-screen editor.
ed [-] [file]	Edits text by line.
edit [-r] file ...	Provides a line editor that is a simplified version of the **ex** editor.
ex [-flags] file	Edits lines interactively, with screen display; creates or uses **file**.
rm [-flag] file ...	Removes files. Use this command with great care.
sed [edit-script(s)-spec] [file]	Edits files in a stream fashion, modifying lines in accordance with an editing script and writing them to standard output.
uniq [-flag] [±num] [infile [outfile]]	Deletes repeated adjacent lines or specified fields in a file and sends output to standard output or to a specified file.
vi -flags [file] ...	Edits using a visual editor based on **ex**; creates or uses each **file** specified.

3.12.4 Comparing and Scanning Files

Command Line	Description
bfs [-] file	Scans large files using a search syntax similar to **ed**.
file [-mreffile] infile **OR** **file -c -mreffile**	Samples a file, determines its file type (ASCII text, data, directory, or whatever), and writes the file type to standard output; or checks the reference file for format errors.
cmp [-flag] file1 file2	Compares two files (usually not text files).
comm -num file1 file2	Finds lines common to two sorted files and selects or rejects them.
bdiff [num] [-s] file1 file2	Finds differences in very large files, using **diff**.
diff [-flags] file1 file2	Compares two files (usually text files).
diff3 [-flag] file1 file2 file3	Compares three files (usually text files).
diffmk [flags] file1 file2 [file3]	Compares **file1** and **file2** and sends output to **file3** or standard output. The output contains formatter requests to make **file1** like **file2** or merely marks the difference between the compared files.
grep [-flags] patternspec [file] ...	Searches file(s) for a pattern and prints each line having the pattern to standard output.
sdiff [-flag] [-w num] [-o outfile] file1 file2	Compares **file1** and **file2** using **diff** and outputs the result to **outfile** or standard output in side-by-side format. You can interactively edit **outfile** as it is output.
sccsdiff -rSID1 -rSID2 [-flags] file [file] ...	Compares two versions of a Source Code Control System file (using **diff**) and optionally prints the difference (using **pr**).
wc [-counttype] file ...	Counts the number of lines, words, and characters in a file.

3.12.5 Merging and Splitting Files

Command Line **Description**

cat file1 >> file2
Appends (concatenates) contents of **file1** to **file2**.

csplit [-flags] file context
Splits files by context. The context is specified as a string pattern, line numbers, or characters recognized by the AIX PS/2 Operating System.

cut fieldlist [file]
Outputs selected fields from each line of a file. The **fieldlist** is a length of field (in characters) or a field specification by imbedded delimiters.

paste [flags] file [file2] ...
Merges lines of several files or subsequent lines of the files to standard output, with delimiters.

sort [flags-and-parameters] [file] ...
Sorts or merges files to standard output.

split num [file] [filesprefix]
Splits a file into pieces of a specified number of lines and stores them to output files of a given prefix, in sequence.

3.12.6 Manipulating Data

Command Line	Description
cal [month] year	Outputs a calendar for a specified month or year, for years 1 through 9999 A.D.
cal [-]	Stores messages by date and outputs messages for current day and next day (including Sunday and Monday, if current day is a Friday or Saturday).
date [+string]	Displays (or sets) current date and time.
e [file [flags]]	Edits text using a full screen editor.
ed [-] [file]	Edits text by line.
ex [parms] [file]	Edits text by line, with screen display.
hyphen [file]	Finds lines that end with a hyphen in input and writes the hyphenated words to standard output.
join [parms] file1 file2	Joins data fields of two files and sends the results to standard output.
spell [parms] file ...	Finds spelling errors and unknown words in input and outputs them to standard output.
tab [-e] [file] ...	Changes space characters into tabs.
tr [-flag] string1 [string2]	Translates characters of a string, deleting or substituting selected characters and writes to standard output.
vi -flags [file] ...	Edits using a visual editor based on **ex**, creates or uses each **file** specified.

3.12.7 Formatting Text

Command Line	Description
cb [-s] [-lineflag] file ...	Formats C-language source code to standard output in an easily read, standard form.
checkcw [delims] [file] ...	Checks a formatter or preprocessor text file for errors in matching **cw** delimiters.
checkmm file ...	Checks a text file for errors in using the memorandum macros and unbalanced equation delimiters.
col [-flags]	Processes text containing half- and reverse-linefeeds to standard output, for printers that cannot perform those movements, and handles multicolumn output instructions. The **col** command can handle **nroff** output that is processed using the **-T37** or **-Tlp** flags.
cw [parms] [file] ...	Prepares constant width text for **troff** and writes it to standard output.
deroff [-flags] file1 [file2]	Removes macros, tokens, and formatter requests from **file1** and outputs the stripped text to **file2** (to standard output, if **file2** not specified). The **flags** select macro packages to strip from the file.
greek -Tterminal	Converts output suitable for a TELETYPE Model 37 workstation to output that uses available features of another workstation. **greek** interprets reverse- and half-line motion control-characters, and characters to output as Greek symbols. Characters that are not output as Greek symbols are simulated by overstriking Roman characters. Special mathematic characters are also supported.
hp [-flags]	Processes input for display on Hewlett-Packard HP2621, HP2640, and HP2645 workstations. Workstation video display enhancements of underlining, half-intensity characters, and inverse video are supported. **hp** can discard all but one of consecutive blank output lines.
hyphen [file]	Finds lines that end with a hyphen in input and writes the hyphenated words to standard output.
m4 [parms] [file] ...	Preprocesses files, substituting macro definitions. **m4** is used primarily for preparing C-language or other programming language source files.
mant [parm] file ...	Formats text using the manual page macros for presenting video display documentation; works like **mmt** command.
mm [parm] file ...	Typesets documents formatted with the memorandum macros,

processing text using **nroff**. **mm** calls **nroff** with the **-h** flag for tab settings. You can also process the memorandum macros by calling **nroff** using the **-mm** flag.

mmt [parm] file ...
Typesets documents formatted with the memorandum macros, processing text using **troff** and, optionally, various preprocessors and postprocessors.

mt [parm] file ...
Typesets documents formatted without using a macro package; works like **mmt** command.

mvt [parm] file ...
Typesets documents using the viewgraph macros, creating output formatted for slide or overhead projector presentations.

newform [[parms] [file]] ...
Changes the format of a text file by lines, by substitution, deletion, and addition of text.

nroff [-flags] file ...
Formats the contents of **file** to standard output. The **flags** specify type of terminal, macro packages to use, and so forth.

ptx [parms] [infile [outfile]]
Generates a permuted index by searching for keywords in input and sorting by line. The output creates a permuted index when processed by **nroff** or **troff**.

stty [flags-and-specs]
Sets, resets, or reports workstation operating parameters.

tab [-e] [file] ...
Changes two or more adjacent space characters into a tab character.

tabs [parms] [-Tterminal] [+m[num]]
Sets tab stops on workstations.

troff [-flags] file ...
Formats the contents of **file** to standard output for phototypesetting. The **flags** specify type of terminal, macro packages to use, and so forth.

tty [-s]
Writes the full path name of your workstation to standard output.

3.12.8 Displaying and Printing Text

Command Line	Description
banner string ...	Writes character strings in large letters to standard output.
cat [-flags] file ...	Concatenates or displays files to standard output.
col [-flags]	Processes text, buffering input having reverse linefeeds and forward and reverse half-linefeeds to standard output.
lp [parms] file	Outputs **file** to a line printer.
nl [parms] [file]	Numbers and outputs lines in a file to standard output.
pg [-flags] [+starting-place] [file] ...	Formats files to the workstation (defined in **TERM**) in display pages.
pr [-flags] [file] ...	Writes a file to standard output, by pages and with a header block.
print [-flags] [file] ...	Enqueues a file for printing to specified or first available device, or cancels printing of a queued file.
prs [parms] file ...	Writes all or part of SCCS file(s) to standard output using keywords to identify data for inclusion.
qdaemon	Schedules jobs enqueued by **print** command.
splp [-flags] [device]	Changes or displays parallel printer driver settings.
tail [value] [-f] [file]	Writes the last part of a file to standard output. The quantity of output may be specified in lines, blocks, or characters, starting from the beginning or end of the file.

3.12.9 Working with Graphs

Command Line	Description
graph [parms]	Prepares a graph from input pairs of x-y coordinate numbers and outputs to standard output in plot file format. The standard output consists of successive points connected by straight lines. Output from **graph** may be used by **tplot** to print the graph; **spline** can preprocess an input file and interpolate intermediate coordinate points before **graph** is run.
spline [parms]	Interpolates a smooth curve of points from input x-y coordinate pairs and outputs (approximately) evenly spaced coordinate pairs.
tplot [-Tterminal] [file]	Produces plotting instructions for a particular workstation from input in plot file format.

3.12.10 Using Shell Commands

Command Line	**Description**
line	Reads one line from standard input and writes it to standard output.
open file [parm]	Opens a virtual terminal and runs a specified file on that terminal.
tee [-flags] file ...	Reads standard input and writes to standard output, copying to a file at the same time.
od [-flags] [[file] [parms]]	Reads specified file or standard input and writes to standard output in a specified format: octal, decimal, hexadecimal, or ASCII.

3.13 Chapter 13. Requests

Subtopics

3.13.1 About This Chapter

This chapter is a reference of instructions (requests, escapes, string registers, keywords, and macros).

3.13.2 Text Appearance

Notes in the following table are interpreted as:

Note	Description
b	The request causes a line break unless prefixed with an ' (acute accent) instead of a period.
D	The parameters of this instruction are part of the current diversion level.
e	The parameters of this instruction are part of the current environment.
F	The request is accepted by **nroff** and **troff**.
M	The instruction is a part of the memorandum macro package.
p	The instruction must be in effect at the time of actual printing to be used.
T	The instruction is used by **troff** and ignored by **nroff**.
W	The request is a call to the **cw** constant-width preprocessor.

Table 13-1. Text Appearance Calls

Call	Notes	Description
.ab [message]	F	Abnormally end formatting. (See "Deliberate Messages" in topic 2.8.18.3.1.)
.af reg format	F	Assign format of number register. (See "Number Registers" in topic 2.8.9.1.)
.CD [parms]	W	Change **cw** delimiters and other **cw** command line options. (See "Constant Width" in topic 2.7.4.11.7.)
.CN [parms]	MW	End constant-width display. (See "Constant Width" in topic 2.7.4.11.7).
.CP [cw][pfnt] [cw][pfnt]...	W	Print in constant width. (See "Constant Width" in topic 2.7.4.11.7.)
.cs fnt[cnum][snum]	Tp	Set constant-width character space mode. (See "Text Adjustment" in topic 2.8.5.2.)
.CW [parms]	MN	Start constant-width display. (See "Constant Width" in topic 2.7.4.11.7.)
.HC [char]	M	Set hyphenation character.
.lg [num]	T	Set ligature mode. (See "Ligatures" in

		topic 2.8.16.3.2.)
.ls num	Fe	Set line spacing. (See "Line Spacing" in topic 2.8.13.1.)
.mk [reg]	FD	Mark current vertical place. (See "Line Drawing and Underlining" in topic 2.8.4.4.)
.ne val	FD	Specify needed vertical space. (See "Reserving Block Space" in topic 2.8.6.2.)
.ns	FD	Turn no-space mode on. (See "No Space Mode" in topic 2.8.13.4.)
.os	F	Output saved vertical distance. (See "Reserving Block Space" in topic 2.8.6.2.)
.PC [pfnt][cw] [pfnt][cw]...	W	Print in constant-width (opposite of **.CP**.) (See "Constant Width" in topic 2.7.4.11.7.)
.ps [[±]num]	Te	Set type size. The initial value is **10** points. When called with no parameter, **.ps** uses the previous point size. (See "Type Size" in topic 2.8.3.3.)
.rs	FD	Restore spacing; turn no-space mode off. (See "No Space Mode" in topic 2.8.13.4.)
.rt [[±]num]	FD	Return to marked vertical place. (See "Line Drawing and Underlining" in topic 2.8.4.4.)
\s[±]num	Te	Set point size. Like the **.ps** request. (See "Type Size" in topic 2.8.3.3.)
.S [±psize] [±vspacing]	M	Set **troff** typesize and vertical spacing. (See "troff Point Size and Vertical Spacing" in topic 2.7.5.5.)
.SM string1 [string2] [string3]	M	Make a string smaller (see "troff Point Size and Vertical Spacing" in topic 2.7.5.5).
.SP [lines]	M	Space vertically. (See "Vertical Spacing" in topic 2.7.5.4.)
.sp [num]	Fb	Space vertical distance. (See "Inserting Vertical Space" in topic 2.8.6.1.)
.ss num	Te	Set space-character size. The spacing is **num/36** em; the initial value is **12/36** em. (See "Text Adjustment" in topic 2.8.5.2.)

.sv num	F	Save vertical distance. (See "Reserving Block Space" in topic 2.8.6.2.)
.vs [num]	Fe	Set vertical baseline spacing. (See "Inserting Vertical Space" in topic 2.8.6.1.)
.tr [a b] [c d]...	F	Translate **a** to **b**, **c** to **d**, and so on. (See "Character Translation for Output" in topic 2.8.16.3.)
.! cmd [args]	F	Execute **cmd** and place its standard output in output at this point. (See "Passing Commands to the System" in topic 2.8.17.)

3.13.3 Fonts

Notes in the following table are interpreted as:

Note	Description
e	The parameters of this instruction are part of the current environment.
F	The request is accepted by **nroff** and **troff**.
M	The macro is a macro of the memorandum macros package, called by an **mm** command or **-mm** flag to a formatter command.
p	The instruction must be in effect at the time of actual printing to be used.
T	The instruction is used by **troff** and ignored by **nroff**.

Table 13-2. Font Calls

Call	Notes	Description
.B [bld] [pfnt] [bld] [pfnt]...	M	Print in bold. (See "Fonts" in topic 2.7.5.2.)
.bd [S] fnt [num]	Fp	Simulate a bold font with overstriking. (See "Emphasized Print" in topic 2.8.3.2.)
.BI [bld] [itl] [bld] [itl] [bld] [itl]	M	Print in bold/italic. (See "Fonts" in topic 2.7.5.2.)
.BR [bld] [rmn] [bld] [rmn]...	M	Print in bold/Roman. (See "Fonts" in topic 2.7.5.2.)
.cu [num]	Fe	Continuous-underline (italicize in **troff**) next **num** input lines. (See "Line Drawing and Underlining" in topic 2.8.4.4.)
\fnum \fc or \f(cc	Te	Change font. (See "Font Selection and Control" in topic 2.8.3.1.)
.fp pnum fnt	T	Mount font. (See "Font Selection and Control" in topic 2.8.3.1.)
.ft [fontchar]	Te	Change font. (See "Font Selection and Control" in topic 2.8.3.1.)
.I [itl] [pfnt] [itl] [pfnt]...	M	Print in italic (underline in **nroff**). (See "Fonts" in topic 2.7.5.2.)
.IB [itl] [bld] [itl] [bld] [itl] [bld]	M	Print in italic/bold. (See "Fonts" in topic 2.7.5.2.)

.IR [itl] [rmn] [itl] [rmn]...	M	Print in italic/Roman. (See "Fonts" in topic 2.7.5.2.)
.R	M	Return to Roman font. (See "Fonts" in topic 2.7.5.2.)
.RB [rmn] [bld] [rmn] [bld]...	M	Print in Roman/bold. (See "Fonts" in topic 2.7.5.2.)
.RI [rmn] [itl] [rmn] [itl]...	M	Print in Roman/italic. (See "Fonts" in topic 2.7.5.2.)
.uf [fnt]	F	Change underline font (used by **.ul**). (See "Line Drawing and Underlining" in topic 2.8.4.4.)
.ul [num]	Fe	Underline (italicize in **troff**) next **num** of input lines. (See "Line Drawing and Underlining" in topic 2.8.4.4.)

3.13.4 Line, Margin, and Indent Control

Notes in the following table are interpreted as:

Note	Description
b	The request causes a page break unless prefaced with an acute accent mark (') instead of a period.
e	The parameters of this instruction are part of the current environment.
F	The request is accepted by **nroff** and **troff**.
m	The default scale indicator is **m**.
M	The macro is a macro of the memorandum macros package, called by an **mm** command or **-mm** flag to a formatter command.

Table 13-3. Line, Margin, and Indent Control Calls

Call	Notes	Description
.ad [char]	Fe	Allow output line adjustment. By default, both margins are adjusted (**.ad b**). (See "Text Adjustment" in topic 2.8.5.2.)
.br	Fb	Break output line fill. (See "Hyphenation and Fill" in topic 2.8.5.1.)
.ce [num]	Fbe	Center the following input text line(s). The **num** is the number of following lines to center, with a default value of **1**. (See "Centered Text Lines" in topic 2.8.5.4.)
.fi	Fbe	Allow output line fill (initially on). (See "Hyphenation and Fill" in topic 2.8.5.1.)
.hc [string]	F	Change hyphenation indicator character. (See "Hyphenation and Fill" in topic 2.8.5.1.)
.hw [word]...	F	Specify hyphenation-exception word list.
.hy [num]	Fe	Allow hyphenation.
.in [[±]num]	Fmbe	Set indent. (See "Margins and Indents" in topic 2.8.13.6.)
.ll [[±]num]	Fme	Set line length.
.mc [char][val]	Fem	Set margin character (and separation space). (See "Margin Character and Line Numbering" in topic 2.8.7.)
.na	Fe	Disallow output line adjustment. (See

		"Text Adjustment" in topic 2.8.5.2.)
.nh	F	Disallow hyphenation. (See "Hyphenation and Fill" in topic 2.8.5.1.)
.nf	Fbe	Disallow output line adjustment and fill. (See "Hyphenation and Fill" in topic 2.8.5.1.)
.nm [[±]val] [num] [znum] [inum]	Fe	Set line numbering mode. (See "Margin Character and Line Numbering" in topic 2.8.7.)
.nn [num]	Fe	Disallow line numbering of the rest of **num** input lines. (See "Margin Character and Line Numbering" in topic 2.8.7.)
.po [[±]val]	Fm	Set page offset. (See "Margins and Indents" in topic 2.8.13.6.)
.SA [parm]	M	Set adjustment (right-margin justification). (See "Right Margin Justification" in topic 2.7.4.2).
.ti [±]num	Fmbe	Temporary indent. The following line is indented relative to the current indent; the total may not be negative. Current indent is unchanged. (See "Temporary Margin Changes and Indenting" in topic 2.8.13.6.1.)

3.13.5 Page Control

Notes in the table below are interpreted as follows:

Note **Description**

b The request causes a page break unless prefaced with an acute accent mark (') instead of a period.

F The request is accepted by **nroff** and **troff**.

M The macro is a macro of the memorandum macros package, called by an **mm** command or **-mm** flag to a formatter command.

u The macro is a ***user exit*** defined by the user, but not usually invoked by the user. Normally, a user exit macro is called by **mm** from inside header, footer, or other macros.

Table 13-4. Page Control Calls

Call	Notes	Description
.1C	M	Specify one-column processing (see "Two-Column Format" in topic 2.7.4.8.1).
.2C	M	Specify two-column processing (see "Two-Column Format" in topic 2.7.4.8.1).
.BE	M	End bottom block (see "Bottom-of-Page Processing" in topic 2.7.4.9).
.bp [[±]num]	Fb	Break page; eject the current page (see "Page Control" in topic 2.8.13.)
.BS	M	Start bottom block (see "Bottom-of-Page Processing" in topic 2.7.4.9).
.EF [parm]	M	Specify even-page footer (see "Even-Page Footer" in topic 2.7.4.5.2).
.EH [parm]	M	Specify even-page header (see "Even-Page Header" in topic 2.7.4.5.1).
.FD [parm] [1]	M	Set footnote default format (see "Format of Footnote Text" in topic 2.7.9.3).
.FE	M	Specify footnote end (see "Delimiting Footnote Text" in topic 2.7.9.2).
.FS [label]	M	Specify footnote start (see "Delimiting Footnote Text" in topic 2.7.9.2).
.OF [parm]	M	Specify odd-page footer (see "Odd-Page Footer" in topic 2.7.4.4.3).
.OH [parm]	M	Specify odd-page header (see "Odd-Page Header" in topic 2.7.4.4.2).

.OP	M	Break to odd page (see "Odd-Page Force" in topic 2.7.4.4.1).
.PF [parm]	M	Specify page footer (see "Page Footer" in topic 2.7.4.3.2).
.PH [parm]	M	Specify page header (see "Page Header" in topic 2.7.4.3.1).
.pl [±]val	F	Set page length (see "Page Control" in topic 2.8.13).
.PM [parm]	M	Set proprietary marking (see "Bottom-of-Page Processing" in topic 2.7.4.9).
.pn [±]num	F	Set page number (see "Page Numbering" in topic 2.8.14.5).
.PX	Mu	Page-header user exit (see "Top-of-Page Processing" in topic 2.7.4.7).
.VM [top] [bottom]	M	Set variable vertical margins (see "Top and Bottom Margin" in topic 2.7.4.1).
.WC [flags]	M	Set width control (see "Two-Column Format" in topic 2.7.4.8.1).

3.13.6 Document Style

Notes in the following table are interpreted as:

Note	Description
e	The parameters of this instruction are part of the current environment.
F	The request is accepted by **nroff** and **troff**.
M	The macro is a macro of the memorandum macros package, called by an **mm** command or **-mm** flag to a formatter command.
u	The macro is a ***user exit*** defined by the user, but not usually invoked by the user. Normally, a user exit macro is called by **mm** from inside header, footer, or other macros.

Table 13-5. Document Style Calls

Call	Notes	Description
.AE	M	End abstract. (See "Abstract and Memorandum for File Cover Sheet" in topic 2.7.3.8.)
.AF [company-name]	M	Specify alternate format of Subject/Date/From block.
.AS [cover] [indent]	M	Start abstract. (See "Abstract and Memorandum for File Cover Sheet" in topic 2.7.3.8.)
.AT [title] ...	M	Specify author's title. (See "Author(s)" in topic 2.7.3.7.)
.AU name [initials] [loc] [dept] [ext] [room] [parm7] [parm8] [parm9]	M	Specify author's information. (See "Author(s)" in topic 2.7.3.7.)
.AV [name]	M	Print approval signature. (See "Approval Signature Line" in topic 2.7.3.10.1.)
.CS [pages] [other] [total] [figs] [tbls] [refs]	M	Print cover sheet. (See "Abstract and Memorandum for File Cover Sheet" in topic 2.7.3.8.)
.fc [delim] [pad]	F	Set field delimiter and pad characters. (See "Field Delimiters" in topic 2.8.16.2.3.)
.FC [closing]	M	Print formal closing. (See "Signature Block" in topic 2.7.3.10.1.)

.lc [char]	eF	Set leader repetition character. (See "Tabs and Leaders" in topic 2.8.16.2.2.)
.lt [[±]num]	eF	Set length of title line. (See "Titles" in topic 2.8.14.1.)
.MT [type] [parm2]	M	Set memorandum type. (See "Memorandum Types and Memorandum Type Field" in topic 2.7.3.3.)
.ND new-date	M	Set new date. (See "Setting the Date" in topic 2.7.3.4.)
.NE	M	End notation. (See "Closing Notations" in topic 2.7.3.10.1.)
.NS [parm]	M	Start notation. (See "Closing Notations" in topic 2.7.3.10.1.)
.OK [keyword]...	M	Specify other keywords for the technical memorandum cover sheet. (See "Other Keys" in topic 2.7.3.9.)
.RF	M	End reference. (See "Delimiting References" in topic 2.7.10.2.)
.RP [parm1] [parm2]	M	Produce reference page. (See "Reference Page" in topic 2.7.10.3.)
.RS [name]	M	Start reference. (See "Delimiting References" in topic 2.7.10.2.)
.SG [parm] [1]	M	Specify signature line. (See "Signature Block" in topic 2.7.3.10.1.)
.ta [pos1] [type1] [pos2] [type2]...	eF	Set tabs. (See "Tabs and Leaders" in topic 2.8.16.2.2.)
.tc [char]	eF	Set tab repetition character. (See "Tabs and Leaders" in topic 2.8.16.2.2.)
.TC [slevel] [spacing] [tlevel] [tab] [hdg1] [hdg2...]	M	Print table of contents. (See "Table of Contents Macro" in topic 2.7.6.3.1.)
.TL	M	Specify title of memorandum. (See "Title" in topic 2.7.3.5.)
.tl 'left'center'right'	F	Print three-part title. (See "Titles" in topic 2.8.14.1.)
.TX	Mu	Table-of-contents user-exit. (See "Table of Contents Macro" in topic 2.7.6.3.1.)
.TY	.Mu	Table-of-contents user-exit (suppresses **CONTENTS** header). (See "Table of Contents Macro" in topic 2.7.6.3.1.)

3.13.7 Document Structures

Notes in the following table are interpreted as:

Note	Description
B	The request is a call to the **tbl** table preprocessor.
F	The request is accepted by **nroff** and **troff**.
M	The macro is a macro of the memorandum macros package, called by an **mm** command or **-mm** flag to a formatter command.
Q	The request is a call to the **eqn/neqn** equation preprocessor.
W	The request is a call to the **cw** constant-width preprocessor.
u	The macro is a ***user exit*** defined by the user, but not usually invoked by the user. Normally, a user exit macro is called by **mm** from inside header, footer, or other macros.

Table 13-6. Document Structure Calls

Call	Notes	Description
.AL [type] [text-indent] [1]	M	Start automatically incremented list (see "Automatically Numbered or Alphabetized Lists" in topic 2.7.8.2.1).
.BL [text-indent] [1]	M	Start bullet list (see "Bullet List and Dash List" in topic 2.7.8.2.2).
.BX parm1 [parm2]	M	Box text (see "Boxed Text" in topic 2.7.5.3).
.CN [parms]	MW	End constant-width display (see "Constant Width" in topic 2.7.4.11.7).
.CW [parms]	MW	Begin constant-width Display (see "Constant Width" in topic 2.7.4.11.7).
.DE	M	End display (see "Static Displays" in topic 2.7.4.11.1 and "Floating Displays" in topic 2.7.4.11.2).
.DF [format] [fill] [right-indent]	M	Start floating display (see "Floating Displays" in topic 2.7.4.11.2).
.DL [text-indent] [1]	M	Start dash list (see "Bullet List and Dash List" in topic 2.7.8.2.2).
.DS [format] [fill] [right-indent]	M	Start static display (see "Static Displays" in topic 2.7.4.11.1).

.EC [title] [override] [flag]	M	Specify equation caption (see "Caption Macros" in topic 2.7.4.11.5).
.EN	MQ	End equation display (see "Equations" in topic 2.7.4.11.4).
.EQ [label]	MQ	Start equation display (see "Equations" in topic 2.7.4.11.4).
.EX [title] [override] [flag]	M	Specify exhibit caption (see "Caption Macros" in topic 2.7.4.11.5).
.FG [title] [override] [flag]	M	Specify figure title (see "Caption Macros" in topic 2.7.4.11.5).
.H level [heading] [subheading]	M	Specify numbered heading (see "Headings" in topic 2.7.6).
.HM [parm1]... [parm7]	M	Set heading mark style (Arabic or Roman numerals or letters) (see "Altering Appearance of Headings" in topic 2.7.6.2).
.HU heading	M	Specify unnumbered heading (see "Headings" in topic 2.7.6).
.HX dlevel rlevel heading	Mu	Heading user-exit X (see "User-Defined Headings" in topic 2.7.6.5).
.HY dlevel rlevel heading	Mu	Heading user-exit Y (see "User-Defined Headings" in topic 2.7.6.5).
.HZ dlevel rlevel heading	Mu	Heading user-exit Z (see "User-Defined Headings" in topic 2.7.6.5).
.LB text-indent mark-indent pad type [mark] [LI-space] [LB-space]	M	Start (begin) list (see "List Begin and Customized Lists" in topic 2.7.8.6).
.LC [list-level]	M	End (clear) several list levels (see "Advanced List Structures" in topic 2.7.8.7).
.LE [1]	M	End list (see "List End" in topic 2.7.8.4).
.LI [mark] [1]	M	Start list item (see "List Items" in topic 2.7.8.3).
.ML mark [text-indent] [1]	M	Start marked list (see "Marked List" in topic 2.7.8.2.3).
.nP	M	Start double-line indented paragraph (see "Numbered Paragraphs" in topic 2.7.7.2).

.P [type]	M	Start paragraph (see "Block and Indented Paragraphs" in topic 2.7.7.1).
.pc char	F	Set page number character (see "Page Number Character" in topic 2.8.14.1.2).
.RL [text-indent][1]	M	Start reference list (see "Reference List" in topic 2.7.8.2.4).
.SK [num]	M	Skip pages (see "Skipping Pages" in topic 2.7.4.10).
.TB [title] [override] [flag]	M	Specify table title (see "Caption Macros" in topic 2.7.4.11.5).
.TE	MB	End table display (see "Tables" in topic 2.7.4.11.3).
.TH [N]	MB	End table header (see "Tables" in topic 2.7.4.11.3).
.TP	Mu	Top-of-page macro (see "Top-of-Page Processing" in topic 2.7.4.7).
.TS [H]	MB	Start table display (see "Tables" in topic 2.7.4.11.3).
.VL text-indent [mark-indent] [1]	M	Start variable-item list (see "Variable-Item List" in topic 2.7.8.2.5).

3.13.8 Input and Output Control

Notes in the following table are interpreted as:

b The request causes a line break unless prefaced with an acute accent mark (') instead of a period.

F The request is accepted by **nroff** and **troff**.

M The macro is a macro of the memorandum macros package, called by an **mm** command or **-mm** flag to a formatter command.

Table 13-7. Input and Output Control Calls

Call	Notes	Description
.ex	F	Exit from the formatter. (See "Insertions From Standard Input" in topic 2.8.18.2.)
.fl	Fb	Flush output buffer. (See "Flushing an Output Buffer" in topic 2.8.18.4.)
.ig char1char2	F	Ignore input until occurrence of **char1char2**. (See "Defining Macros with Parameters" in topic 2.8.8.3.)
.nx file	F	Switch input file. (See "File Switching and Piping" in topic 2.8.18.1.)
.pi program	N	Pipe output to a program. (See "File Switching and Piping" in topic 2.8.18.1.)
.pm [t]	F	Print macro names and sizes. (See "Deliberate Messages" in topic 2.8.18.3.1.)
.RD [prompt] [diversion] [string]	M	Read insertion from terminal.
.rd [prompt]	F	Read insertion from standard input. (See "Insertions From Standard Input" in topic 2.8.18.2.)
.so file	F	Switch input files; afterwards, come back to current input file. (See "File Switching and Piping" in topic 2.8.18.1.)
.tm [string]	F	Print message on terminal. (See "Deliberate Messages" in topic 2.8.18.3.1.)

3.13.9 Formatter Programming Functions

Notes in the following table are interpreted as:

D The parameters of this instruction are part of the current diversion level.

e The parameters of this instruction are part of the current environment.

F The request is accepted by **nroff** and **troff**.

Table 13-8. Formatter Programming Function Calls

Call	Notes	Description
.am name [end]	F	Append to a macro. (See "Strings and Macros" in topic 2.8.8.)
.as name string	F	Append to defined string. (See "Strings and Macros" in topic 2.8.8.)
.c2 [char]	eF	Set no-break control character. (See "Request Line Prefixes" in topic 2.8.16.2.1.)
.cc [char]	eF	Set control character. (See "Request Line Prefixes" in topic 2.8.16.2.1.)
.ch macro [-]num	F	Change trap location. (See "Traps" in topic 2.8.11.2.)
.da [name]	DF	Divert and append to specified macro. (See "Strings and Macros" in topic 2.8.8.)
.de name [end]	F	Define or redefine macro. (See "Defining Simple Macros" in topic 2.8.8.2.)
.di [xy]	DF	Divert output to specified macro. (See "Diversions" in topic 2.8.11.)
.ds name string	F	Define a string. (See "Defining Strings" in topic 2.8.8.1.)
.dt num [macro]	DF	Set diversion trap. (See "Traps" in topic 2.8.11.2.)
.ec [char]	F	Set escape character. (See "Escape Requests" in topic 2.8.16.2.)
.el parms	F	Specify else portion of an if/else condition. (See "Conditional Tests" in topic 2.8.10.)
.em macro	F	Specify end macro. (See "Traps" in topic 2.8.11.2.)

.eo	F	Turn off escape character mechanism. (See "Escape Requests" in topic 2.8.16.2.)
.ev [num]	F	Environment switch (by push down). (See "Environment Switching" in topic 2.8.12.)
.ie parms	F	Specify the if portion of an if/else conditional. (See "Conditional Tests" in topic 2.8.10.)
.if parms	F	Specify if conditional. (See "Conditional Tests" in topic 2.8.10.)
.it num [macro]	eF	Set input-line count-trap. (See "Traps" in topic 2.8.11.2.)
.nr reg num [[±]increment]	F	Set number register. Autoincrement level may be set also. (See "Number Registers" in topic 2.8.9.1.)
.rm name	F	Remove request, macro, or string register. (See "Defining Simple Macros" in topic 2.8.8.2.)
.rn name new	F	Rename request, macro, or string register. (See "Strings and Macros" in topic 2.8.8.)
.rr numreg	F	Remove register specified. (See "Number Registers" in topic 2.8.9.1.)
.wh [±]num macro	F	Where specified, set trap. (See "Traps" in topic 2.8.11.2.)

3.14 Chapter 14. Registers

Subtopics

3.14.1 About This Chapter

A ***register*** is an allocated address in memory to store data. Most registers used by the formatter are ***number registers***; data is stored as a numeric value. Other registers store strings of ASCII text and are called ***string registers***.

3.14.2 Number Registers

Any register having a single-character name may be set from the command line. The lowercase alphabetic single-character registers are reserved for user definition. The following registers are predefined number registers. The interpretations for **Notes** are:

Note **Description**

F The register is used by both **nroff** and **troff** formatter.

M The register is defined by the memorandum macro package (MM). Unless otherwise indicated, it may be used by both **troff** and **nroff**.

U The register is not used unless some reserved user-defined macro is defined.

T The register is used by **troff** but not by **nroff**.

N The register is used by **nroff** but not by **troff**.

r The register is read-only; you cannot alter the values contained in it using the number register request, **.nr**.

s The register **must** be set from the command line or before the MM definitions are read by the formatter (See "Setting Number Registers from the Command Line" in topic 2.7.2.4.5, and "Initialization Files" in topic 2.7.2.4.6).

Description contains, for each register, a brief description, section reference, initial (default) value, and the legal range of values. The initial value is given in boldface within the displayed range.

Table 14-1. Number Registers

Name	**Notes**	**Description**
A	Ms	Setting for preprinted forms. Value from **0** - **2**. (See "Setting Number Registers from the Command Line" in topic 2.7.2.4.5.)
Au	M	Setting for inhibiting output of author's location, department, room, and extension in "from" portion of a memorandum (on first page). Value from **0** - **1**. (See "Author(s)" in topic 2.7.3.7.)
C	Ms	Copy type (**Original**, **DRAFT**, and so on). Value from **0** (Original) - **4**. (See "Setting Number Registers from the Command Line" in topic 2.7.2.4.5.)
Cl	M	Contents level; level of headings saved for table of contents. Value from **0** -**2** - **7**.
Cp	M	Placement of lists of figures, tables, equations, and so on. Value from **0** - **1**. (See "Constant Width" in topic 2.7.4.11.7 and "Contents Registers" in

		topic 2.7.6.3.2.)
ct	F	Character type (set by width escape request).
D	Ms	Debug flag. Value from **0** (debug mode off) - **1**. (See "Setting Number Registers from the Command Line" in topic 2.7.2.4.5.)
De	M	Display eject register for floating displays. Value from **0** - **1**. (See "Floating Displays" in topic 2.7.4.11.2.)
Df	M	Display format register for floating displays. Value from **0** - **5**. (See "Floating Displays" in topic 2.7.4.11.2.)
dl	F	Width (maximum) of last completed diversion.
dn	F	Height (vertical size) of last completed diversion.
Ds	M	Static display pre- and post-space. Value from **0** - **1**. (See "Static Displays" in topic 2.7.4.11.1.)
dw	F	Current day of the week (1=Sunday, ..., 7=Saturday).
dy	F	Current day of the month (1-31).
E	Ms	Font control for the Subject/Date/From memorandum fields. Value of **0** (the **nroff** default, normal) or **1** (the **troff** default, italic). (See "Setting Number Registers from the Command Line" in topic 2.7.2.4.5.)
Ec	M	Equation counter used by **.EC** macro (incremented by **1** for each **.EC** call). Value is **0** or greater. (See "Caption Macros" in topic 2.7.4.11.5.)
Ej	M	Eject-page flag for headings. Value from **0** - **7**; the **0** default results in no page eject for headings. (See "Altering Appearance of Headings" in topic 2.7.6.2.)
Eq	M	Equation label placement. Value from **0** - **1**; the **0** default results in the label being right-adjusted.
Ex	M	Exhibit counter used by **.EX** macro. Value is **0** or greater (incremented by **1** for each **.EX** call). (See "Caption Macros" in topic 2.7.4.11.5.)
Fg	M	Figure counter used by **.FG** macro. Value of **0** or greater (incremented by **1** for each **.FG** call). (See "Caption Macros" in topic 2.7.4.11.5.)
Fs	M	Footnote separation (space between footnotes). Value from **0**-**1** or greater. (See "Spacing between Footnote Entries" in topic 2.7.9.4.)
H1-H7	M	Heading counters for heading levels 1-7. Value of **0** or greater (incremented by **.H** of the corresponding level or by **.HU** if at level

		corresponding to value given by register **Hu**. (Registers **H2-H7** are reset to **0** by any heading at a lower-numbered level.) (See "Altering Appearance of Headings" in topic 2.7.6.2.)
Hb	M	Heading break level above which there is a break (after **.H** and **.HU**). Value from **0** - **2** - **7**. (See "Altering Appearance of Headings" in topic 2.7.6.2.)
Hc	M	Heading centering level for **.H** and **.HU** (the level below which the heading is centered). Value from **0** - **7**. A value of **0** results in no centered headings. (See "Altering Appearance of Headings" in topic 2.7.6.2.)
Hi	M	Heading temporary indent for stand-alone headings (after **.H** and **.HU**). Value from **0** - **1** - **2**. A value of **1** causes an indent as for paragraph. (See "Altering Appearance of Headings" in topic 2.7.6.2.)
hp	F	Current horizontal place on input line.
Hs	M	Heading space level below which a blank line is added. Value from **0** - **2** - **7**. A value of **2** causes a space only after **.H 1** or **.H 2**. (See "Altering Appearance of Headings" in topic 2.7.6.2.)
Ht	M	Heading type: single or concatenated numbers for numbered headers. Value from **0** - **1**. A value of **0** results in concatenated header numbers. (See "Altering Appearance of Headings" in topic 2.7.6.2.)
Hu	M	Heading level for unnumbered heading. Value from **0** - **2** - **7**. (See "Headings" in topic 2.7.6.)
Hy	M	Hyphenation control for body of document. Value from **0** - **1**. A value of **0** causes automatic hyphenation to be off. (See "Hyphenation" in topic 2.7.2.5.4.)
L	Ms	Length of page. In **nroff**, the value may be **20** - **66** or greater. In **troff**, the value may be **2i** - **11i** or greater. The default for **nroff** is 66 lines; for **troff**, it is 11 inches. (See "Setting Number Registers from the Command Line" in topic 2.7.2.4.5.)
Le	M	List of Equations. Value from **0** - **1**. If value is **0**, a list is not produced. (See "Lists of Figures, Tables, Exhibits, and Equations" in topic 2.7.4.11.6.)
Lf	M	List of Figures. Value from **0** - **1**. If value is **1**, a list is produced. (See "Lists of Figures, Tables, Exhibits, and Equations" in topic 2.7.4.11.6.)
Li	M	List indents. In **nroff**: value of **0** - **6** or greater. In **troff**: **0** - **5** or greater. (See "Automatically Numbered or Alphabetized Lists" in topic 2.7.8.2.1)

ln	F	Output line number.
Ls	M	List spacing between items by level value from **0** - **6**. A value of **6** causes spacing between all levels.
Lt	M	List of tables. Value from **0** - **1**. A list is produced if the value is **1** (See "Lists of Figures, Tables, Exhibits, and Equations" in topic 2.7.4.11.6.)
Lx	M	List of exhibits. Value from **0** - **1** . A list is produced if the value is **1**. (See "Lists of Figures, Tables, Exhibits, and Equations" in topic 2.7.4.11.6.)
mo	F	Current month (1-12).
N	Ms	Numbering style. Value from **0** - **5**. (See "Setting Number Registers from the Command Line" in topic 2.7.2.4.5.)
nl	F	Vertical position of last printed text baseline.
Np	M	Numbering style for paragraphs. Value from **0** - **1**. A **0** value causes unnumbered paragraphs. (See "Numbered Paragraphs" in topic 2.7.7.2.)
O	Ms	Offset of page. In **nroff**, value from **0**. - **.75i** or greater. In **troff**, value from **0i** - **0.5i** or greater. (See "Setting Number Registers from the Command Line" in topic 2.7.2.4.5.)
Oc	M	Table of contents page numbering style. Value from **0** - **1**. A **0** value causes lowercase Roman page-numbering for table of contents. (See "Contents Registers" in topic 2.7.6.3.2.)
Of	M	Figure caption style. Value from **0** - **1**. A value of **0** results in a period as separator. (See "Caption Macros" in topic 2.7.4.11.5.)
P	M	Page number, managed by **mm**. Value of **0** or greater. (See "Setting Number Registers from the Command Line" in topic 2.7.2.4.5.)
Pi	M	Paragraph indent. For **troff**, value from **0** - **3** or greater. For **nroff**, value from **0** - **5** or greater. (See "Block and Indented Paragraphs" in topic 2.7.7.1.)
Ps	M	Paragraph spacing. Value from **0** - **1** or greater. A value of **1** causes one blank vertical space between paragraphs. (See "Paragraph Separation" in topic 2.7.7.3.)
Pt	M	Paragraph type. Value from **0** - **2**. A value of **0** sets left-justified paragraphs. (See "Block and Indented Paragraphs" in topic 2.7.7.1.)

Pv	M	Handles **PRIVATE** header string. Value from **0** - **2**. A **0** value means the string is not printed. (See "Top-of-Page Processing" in topic 2.7.4.7.)
Rf	M	Reference counter used by **.RS** call. Value **0** or greater, incremented by **1** for each **.RS** call. (See topic 2.7.10.1.)
S	Ms	Default typesize (**troff only**). Value from **6** - **10** - **36**. (See "Setting Number Registers from the Command Line" in topic 2.7.2.4.5.)
sb	F	Depth of string below baseline (generated by width escape request).
Si	M	Standard display indent. For **troff**, value from **0** - **3** or greater. For **nroff**, **0** - 5 or greater. (See "Static Displays" in topic 2.7.4.11.1.)
st	F	Height of string above base line (generated by width escape request).
T	Ms	Type of output device (**nroff only**). Value from **0** - **2**. (See "Setting Number Registers from the Command Line" in topic 2.7.2.4.5.)
Tb	M	Table counter. Value **0** or greater, incremented by **1** for each **.TB** call. (See "Caption Macros" in topic 2.7.4.11.5 and "Tables" in topic 2.7.4.11.3.)
U	Ms	Underlining style (**nroff** only) for headers. Value from **0** - **1**. A value of **0** causes a continuous underline when the physical device can support it. (See "Setting Number Registers from the Command Line" in topic 2.7.2.4.5.)
W	Ms	Width of page (line and title length). In **troff**: **2i** - **6i** - **7.54i**. In **nroff**: **10** - (**6i**) - **1365**. (See "Setting Number Registers from the Command Line" in topic 2.7.2.4.5.)
yr	F	Last two digits of current year.
:g	M	Current list level. Each call to a list-start macro increments **:g** and each **.LE** decrements it. (See topic 2.7.8.7.)
:R	M	Reference Select. (See "Reference Page" in topic 2.7.10.3.)
;0	MU	Space following heading, when **.HX** is defined. (See topic 2.7.6.5.)
;3	MU	Adjustment factor for space needed before heading is printed, when **.HX** is defined. (See topic 2.7.6.5.)
.$	Fr	Number of parameters available at the current macro level.

.A	Fr	Set 1 in **troff**, if -a option used; always 1 in **nroff**.
.a	Fr	Postline extra line-space most recently utilized using \x'**num**'.
.b	N	Emboldening factor of the current font used by **.bd** and by the **-u** flag.
.c	Fr	Number of lines read from current input file, including .so files.
.d	Fr	Current vertical place in current diversion; equal to nl, if no diversion.
.F	Fr	A string that is the name of the current input file.
.f	Fr	Current font as physical quadrant (1-4).
.H	Fr	Available horizontal resolution in basic units.
.h	Fr	Text baseline high-water mark on current page or diversion.
.i	Fr	Current indent.
.j	Fr	Current adjustment mode and type indicator. Can be saved and used later as a parameter to **.ad** request to restore a previous adjustment.
.k	Fr	Horizontal size of the current partially collected output line in the current environment.
.L	Fr	Current line spacing; the value of the most recent **.ls** request.
.l	Fr	Current line length.
.n	Fr	Length of text portion on previous output line.
.o	Fr	Current page offset.
.P	Fr	Contains value of **1** if current page is printing, **0** otherwise (if, for example, the current page does not appear in the **-o** flag list of the command line).
.p	Fr	Current page length.
.R	Fr	Count of available number registers for use.
.s	Fr	Current type size.
.T	Fr	Set to 1 in **nroff**, if -T option used; always **0** in **troff**.
.t	Fr	Distance to the next trap.
.u	Fr	Equal to 1 in fill mode and **0** in no-fill mode.
.V	Fr	Available vertical resolution in basic units.

.v	Fr	Current vertical line spacing.
.w	Fr	Width of previous character.
.x	Fr	Reserved version-dependent register.
.y	Fr	Reserved version-dependent register.
.z	Fr	Name of current diversion.

(1) For **nroff**, these values are **unscaled** numbers representing lines or character positions; for **troff**, these values must be **scaled**.

3.14.3 String Registers

A string register is defined by the **ds** request and often is a sequence of formatter requests that are read when the string register is called. The following string registers are predefined. The interpretations for **Notes** are:

Note	Description
F	The register is used by both **nroff** and **troff** formatter.
M	The register is defined by the memorandum macro package (MM). Unless otherwise indicated, it can be used by both **troff** and **nroff**.
r	The register is read-only; you cannot change the value stored in it.
U	The register is not used unless some reserved user-defined macro is defined.
T	The register is used by **troff** but not by **nroff**.
N	The register is used by **nroff** but not by **troff**.

Table 14-2. String Registers

Name	Notes	Description
BU	M	A bullet string called by the bullet token. is generated in **troff**, and in **nroff** an o is overstruck by a plus sign. (See "Bullets" in topic 2.7.2.5.7.)
Ci	M	Table-of-contents indent list, up to seven scaled parameters for heading levels. (See "Contents Indent Register" in topic 2.7.6.3.3.)
DT	M	Date uses current date, unless overridden, of the form **Month dd, yyyy** (**April 20, 1989**, for example). (See "Setting the Date" in topic 2.7.3.4.)
EM	M	Em dash string. In **troff** an em-long dash is generated, and in **nroff** two en-dashes are generated. (See "Dashes, Minus Signs, and Hyphens" in topic 2.7.2.5.5.)
F	M	Footnote numberer. (See "Automatic Numbering of Footnotes" in topic 2.7.9.1.)
HF	M	Heading-level font string, up to seven codes for heading levels 1 through 7. Initial values are **3322222**. Levels 1 and 2 are bold; levels 3-7 are underlined in **nroff** and italic in **troff**. (See "Altering Appearance of Headings" in topic 2.7.6.2.)
HP	MT	Heading type size list, which takes up to seven absolute or relative type size codes for heading

		levels 1 through 7. For **troff** use only. (See "Altering Appearance of Headings" in topic 2.7.6.2.)
In	M	Company name, default value is **<<Company Name>>**. (See topic 2.7.4.9.)
Le	M	Title for list of equations. (See "Lists of Figures, Tables, Exhibits, and Equations" in topic 2.7.4.11.6.)
Lf	M	Title for list of figures. (See "Lists of Figures, Tables, Exhibits, and Equations" in topic 2.7.4.11.6.)
Lt	M	Title for list of tables. (See "Lists of Figures, Tables, Exhibits, and Equations" in topic 2.7.4.11.6.)
Lx	M	Title for list of exhibits. (See "Lists of Figures, Tables, Exhibits, and Equations" in topic 2.7.4.11.6.)
Pv	M	"Private" header. (See "Top-of-Page Processing" in topic 2.7.4.7.)
RE	M	SCCS data for the memorandum macro package. Initial value is **release.revision** of current MM package. For example, **3.5**, if the current memorandum macro release is three and the revision level is five. (See topic 2.7.4.3.1.)
Rf	M	Reference numberer. (See "Reference Page" in topic 2.7.10.3.)
Rp	M	Title for references. (See "Reference Page" in topic 2.7.10.3.)
T	M	Tab character.
Tm	M	Trademark string. **troff** places the letters **TM** in reduced typesize, one half-line above the text that it follows ((TM)), and **nroff** handles the letters in a machine-dependent way. (See "The Trademark" in topic 2.7.2.5.8.)
U	M	The short name of the operating system, for example, **AIX PS/2**.
UF	M	The full name of the system, for example, **IBM AIX PS/2 Operating System**.
'	M	Acute accent. The formatter places the mark centered over the character that it follows at input. (See "Character Accents" in topic 2.7.5.1.)
,	M	Cedilla accent. The formatter places the "tail" mark centered and attached beneath the character that it follows at input. (See "Character Accents" in topic 2.7.5.1.)

(^)	M	Circumflex accent. The formatter places the mark centered over the character that it follows at input. (See "Character Accents" in topic 2.7.5.1.)
`	M	Grave accent. The formatter places the mark centered above the character it follows at input. (See "Character Accents" in topic 2.7.5.1.)
(~)	M	Tilde accent. The formatter places the mark centered above the character it follows at input. (See "Character Accents" in topic 2.7.5.1.)
:	M	Umlaut accent, lowercase. The formatter places the umlaut mark ((..)) centered above the character it follows at input. (See "Character Accents" in topic 2.7.5.1.)
;	M	Umlaut accent, uppercase. The formatter places an umlaut centered above the character it follows at input and locates it above the line to accommodate an uppercase letter. (See "Character Accents" in topic 2.7.5.1.)
}0	MU	Can contain heading mark string when **.HX** is defined. (See topic 2.7.6.5.)
}2	MU	Can contain two unpaddable spaces to separate a run-in heading from text when **.HX** is defined. (See topic 2.7.6.5.)
.F	Fr	A string that is the name of the current input file.

A.0 Appendix A. Escape Sequences

Escape sequences are formatter functions that serve as tokens that call the contents of string (or number) registers, or serve as a form of request that can be used any place on a line of input.

In the following table, **Notes** mean:

c Sequence is interpreted in copy mode.

F Sequence is used by **nroff** and **troff**.

T Sequence is used by **troff** and ignored by **nroff**.

Table A-1. Escape Sequences

Call	Notes	Description
****	cF	\ (to prevent or delay the interpretation of \).
\¦	F	¦(acute accent); equivalent to **\(aa**.
\`	F	`(grave accent); equivalent to **\(ga**.
\-	F	- Minus sign in the current font.
\.	cF	Period (dot).
****(space)	F	Unpaddable space-size space character.
\0	F	Digit-width space.
|	F	1/6 em narrow space character (zero width in **nroff**).
\^	F	1/12 em half-narrow space character (zero width in **nroff**).
\&	F	Zero-width character (nonprinting character).
\!	F	Transparent line indicator.
\"	cF	Beginning of comment ignored by formatter.
\$num	cF	Macro parameter 1=**num**=9.
\%	F	Default optional hyphenation character.
\(xx	F	Special character named **xx**.
***x**	cF	String register **x**.
***(xx**	cF	String register **xx**.
\a	cF	Non-interpreted leader character.
\b'string'	F	Bracket building function.

\c	F	Interrupt text processing (continue word across input line-break).
\d	F	Forward (down) 1/2 em vertical motion (1/2 line in **nroff**).
\e	F	Printable version of the current escape character.
\fx	F	Change to font named **x**.
\f(xx	F	Change to font named **xx**.
\fnum	F	Change to font mount position **num**.
\gx **OR** **\g(xx**	F	Return format of register **x** or **xx**; returns nothing if **x** or **xx** is not yet referenced. (Useful with **.af**.)
\jx **OR** **\j(xx**	F	Mark in register **x** or **xx** the current horizontal position on the output line.
\h'val'	F	Local horizontal motion; move **val** distance (negative value is move left, positive is move right).
\kx	F	Mark horizontal input place in register **x**.
\l'val'	F	Horizontal line drawing function. **val** may have a scale indicator.
\L'val'	F	Vertical line drawing function. **val** may have a scale indicator.
\nx	cF	Number register **x**.
\n(xx	cF	Number register **xx**.
\o'string'	F	Overstrike characters in **string**
\p	F	Break and proportionally-space output line across length of line.
\r	F	Backscroll one em (one line in **nroff**).
\snum **OR** **\s±num**	F	Point-size change function.
\t	cF	Noninterpreted horizontal tab.
\u	F	Backscroll a half em in **troff**, half line in **nroff**.
\v'val'	T	Local downward vertical motion **val** distance.
\w'string'	F	Width of **string**.

\x'val'	F	Extra line-space function (negative **val** before, positive after).
\zchar	F	Print **char** with zero width (without spacing).
\{	F	Begin conditional input.
\}	F	End conditional input.
\(new-line)	cF	Concealed (ignored) new-line, read following line as continuation of previous.
\X	F	**X** is any character not listed above.

B.0 Appendix B. Keywords

The equation preprocessor, **eqn**, uses keywords as instructions, converting them to formatter requests. The table preprocessor, **tbl**, uses keywords as global parameters to the **.TS** table-start request. **tbl** also uses keyletters and keyletter parameters to set the table format.

In the following table, the **Notes** mean:

B The call is a **tbl** request.

Bw The call is a **tbl** global option.

Bd The call is a **tbl** data request.

Bl The call is a **tbl** keyletter or keyletter parameter that specifies a table column or row format or a data field feature.

Qk The call is an **eqn** and **neqn** keyword (except as noted).

Qc The call is an **eqn** and **neqn** character call.

Table B-1. Keywords

Call	Notes	Description
A OR a	Bl	Alphabetic sub-column (see topic 2.10.2.3).
elm1 above elm2	Qk	Place elements, **elm1** over **elm2** (see topics 2.9.17 and 2.9.18).
allbox	Bw	Enclose each item in the table in a box. Each row is separated by a horizontal line, and each column by a vertical line, and the whole table is boxed (see topic 2.10.2.2).
alpha	Qc	&alpha. equation (Greek) character.
approx	Qc	&app. equation symbol.
keyB **OR** **keyb**	Bl	Bold font. A keyletter parameter that specifies bold font for a column. (Equivalent to **keyFB**, **keyFb**, **keyfB**, and **keyfb**.)
back num	Qk	Backward local motion. Within a line, move left by the amount **num**. The amount of motion is **num**/100 em (see topic 2.9.21).
x bar	Qk	Places a horizontal (macron) line above **x** (see topic 2.9.10.4).
beta	Qc	ß equation (Greek) character.
bold x y [**X z**] ...	Qk	Bold. Make character or element **x** bold font (see topic 2.9.11).

box	Bw	Enclose the table in a box (see topic 2.10.2.2).
C OR c	Bl	Centered column (see topic 2.10.2.3).
ccol	Qk	Centered column of a matrix (see topic 2.9.18).
cdot	Qc	(.) equation symbol.
center	Bw	Center table horizontally in (offset and margined) page (see topic 2.10.2.2).
chi	Qc	&chi. equation (Greek) character.
col	Qk	(see topic 2.9.18.)
cpile	Qk	Center-aligned pile (see topic 2.9.17).
define xy 'string'	Qk	Define **xy** as **string** (see topic 2.9.8.2).
delim (**xy**)	BwQk	Set expression delimiters; **x** is the opening delimiter, and **y** is the closing delimiter. (For **eqn** and **neqn** use, see topic 2.9.5; for **tbl** use, see topic 2.10.2.2.)
del	Qc	&del. equation symbol.
DELTA	Qc	&Delta. equation (Greek) character.
delta	Qc	&delta. equation (Greek) character.
x dot	Qk	Dot accent above **x** (see topic 2.9.10.4).
x dotdot	Qk	Double dot (umlaut) accent over **x** (see topic 2.9.10.4).
doublebox	Bw	Enclose the table in two boxes (see topic 2.10.2.2).
down num	Qk	Adjust vertical line placement downward in **num**/100 em (see topic 2.9.21).
x dyad	Qk	( ) Dyad diacritical mark above **x** (left-right arrow above character) (see topic 2.9.10.4).
keyE **OR** **keye**	Bl	Equal width columns. All columns with a keyletter that is suffixed by an **e** or **E** are made the same width (see topic 2.10.2.3).
epsilon	Qc	&epsilon. equation (Greek) character.
eta	Qc	&eta. equation (Greek) character.
expand	Bw	Make the table as wide as the current line length (see topic 2.10.2.2).

keyFfnt **OR** **keyffnt**	Bl	Font specifier. A keyletter parameter that specifies a font to use. **fnt** may be a recognized font specification or a **troff** typesetter mount position number (see topic 2.10.2.3).
fat elm	Qk	Emphasized print of element **elm** (see topic 2.9.11).
font x	Qk	Use font **x** (see topic 2.9.11).
bigsym from elm1 [to elm2]	Qk	Plex equation construction keyword places **elm1** element over **bigsym** symbol, which may be a &sum., &integral., &product., &union., or &intersect. (see topic 2.9.20).
fwd num	Qk	Local motion to the right in **num**/100 em (see page 2.9.21).
GAMMA	Qc	&Gamma. equation (Greek) character.
gamma	Qc	&gamma. equation (Greek) character.
gfont fnt	Qk	Global font specifier for all equations in a document; use **fnt** specified (see topic 2.9.11).
grad	Qc	&del. equation symbol.
gsize val	Qk	Global typesize specifier for all equations in a document; use a legal **val** typesize or an increment or decrement of current typesize (see topic 2.9.11).
x hat	Qk	Circumflex accent over **x** (see topic 2.9.10.4).
half	Qc	½ equation symbol.
keyI **OR** **keyi**	Bl	Italic font. A keyletter parameter that specifies italic font for a column. The same as **keyFI**, ...
inf	Qc	&infinity. equation symbol.
int	Qc	&integral. equation symbol.
inter	Qc	&intersect. (intersection) equation symbol.
iota	Qc	&iota. equation (Greek) character.
italic elm	Qk	Print the **elm** in italic font (see topic 2.9.11).
kappa	Qc	&kappa. equation (Greek) character.
L OR l	Bl	Left-adjusted column (see topic

		2.10.2.3).
LAMBDA	Qc	&Lambda. equation (Greek) character.
lambda	Qc	&lambda. equation (Greek) character.
lcol elm	Qk	Place the **elm** of a matrix in a left-aligned column. There must be a preceding **matrix** keyword (see topic 2.9.18).
left c construct	Qk	Build a large construction character **c** (bracket, brace, parenthesis, or vertical bar) and left-align **construct** on it (see topic 2.9.19).
linesize (**num**)	Bw	Set lines or rules, for example, for **box**, in **num**-point type (see topic 2.10.2.2).
... **[elm0] lineup elm1** ...	Qk	Align construct so **elm1** aligns under a previous **mark** (see topic 2.9.15).
lpile elm1 elm2 ...	Qk	Build a vertically piled, left-aligned construct with **elm1** over **elm2** (see topic 2.9.17).
... **[elm0] mark elm1** ...	Qk	Set the **elm1** mark to line up a subsequent construct on (see topic 2.9.15).
matrix { **column-elms** . . . **}**	Qk	Build a matrix as an array of columns (see topic 2.9.18).
mu	Qc	¦ equation (Greek) character.
N OR n	Bl	Numeric column (see topic 2.10.2.3).
ndefinetk 'string'	Qk	define an **nroff** token **tk** as holding **string** (see topic 2.9.8.2). Used with **neqn** only.
nothing	Qc	Prints nothing; equation empty place-holder symbol. (Same as a pair of double quotation marks.)
nu	Qc	&nu. equation (Greek) character.
OMEGA	Qc	&Omega. equation (Greek) character.
omega	Qc	&omega. equation (Greek) character.
omicron	Qc	&omicron. equation (Greek) character.
elm1 over elm2 [elm3] ...	Qk	Create a fraction construction; **elm1** is placed above **elm2** with a separating

		horizontal bar; **elm3** is aligned with the dividing bar (see topic 2.9.12).
keyPval OR keypval	Bl	Point size specifier. A keyletter parameter that specifies the type size for corresponding table entries (for **troff**). **val** is an absolute point size, or a signed digit indicating an increment or decrement (up to 9 points) of the current point size (see topic 2.10.2.3).
partial	Qc	&partial. equation symbol.
PHI	Qc	&Phi. equation (Greek) character.
phi	Qc	&phi. equation (Greek) character.
PI	Qc	&Pi. equation (Greek) character.
pi	Qc	&pi. equation (Greek) character.
pile elm1 elm2 ...	Qk	Make a construction, placing **elm1** over **elm2** (see topic 2.9.17).
prime	Qc	' equation symbol.
prod	Qc	&product. (product) equation symbol.
PSI	Qc	&Psi. equation (Greek) character.
psi	Qc	&psi. equation (Greek) character.
R OR r	Bl	Right-adjusted column.
\Rx	Bd	Repeat table entry character **x** as many times as necessary to fill the data field the width of a column.
rcol elm [elm] ...	Qk	Create a right-aligned column for the preceding **matrix** call (see topic 2.9.18).
rho	Qc	&rho. equation (Greek) character.
right c construct	Qk	Build a large construction character **c** (bracket, brace, parenthesis, or vertical bar) and right-align **construct** on it (see topic 2.9.19).
roman elm	Qk	Print element **elm** in Roman font (see topic 2.9.11).
rpile elm1 elm2 ...	Qk	Build a right-aligned pile construct (see topic 2.9.17).
S OR s	Bl	Horizontally spanned column heading (see topic 2.10.2.3).
SIGMA	Qc	&Sigma. equation (Greek) character.
sigma	Qc	&sigma. equation (Greek) character.

size val elm	Qk	Set **elm** to type size **val**; val can be any valid type size or an increment or decrement of the current typesize (see topic 2.9.11) .
sqrt elm	Qk	Square root of **elm** (see topic 2.9.13).
elm1 sub elm2	Qk	Subscript; **elm2** is subscripted and attached to **elm1** (see topic 2.9.14).
sum	Qc	&sum. equation summation symbol (see topic 2.9.20).
elm1 sup elm2	Qk	Subscript; **elm2** is subscripted and attached to **elm1** (see topic 2.9.11).
keyT **OR** **keyt**	Bl	Vertically spanning at the top.
T{	Bd	Begin text block as table data entry.
}T	Bd	End text block as table data entry.
.T&	B	Request to continue table with new parameters.
tab (**x**)	Bw	Set the data separation character to **x** ASCII TAB is the default (see topic 2.10.2.2).
tau	Qc	&tau. equation (Greek) character.
tdefine tk 'string'	Qk	Define **troff** token **tk** to hold **string** (see topic 2.9.8.2). Used by **eqn** only.
.TE	B	Table-end request.
THETA	Qc	&Theta. equation (Greek) character.
theta	Qc	&theta. equation (Greek) character.
n tilde	Qk	Put a (**~**) (tilde) over the character **n** (see topic 2.9.10.4).
times	Qc	¦ equation multiplication symbol.
bigsym [from elm1] to elm2	Qk	Plex equation construction keyword places **elm2** element under **bigsym** symbol (see topic 2.9.20).
.TS	B	Table-start request.
keyU **OR** **keyu**	Bl	Stagger columns. A keyletter followed by a **U** or **u** causes an entry to be moved up half a line from a preceding entry. Staggered columns cannot be used with an **allbox** keyword specification for a table and may not work with some printers (see

		topic 2.10.2.3).
x under	Qk	Place a character-wide underscore mark under **x** (see topic 2.9.10.4).
union	Qc	&union. equation symbol.
up num	Qk	Local motion upward by **num**/100 em (see topic 2.9.21).
UPSILON	Qc	&Upsilon. equation (Greek) character.
upsilon	Qc	&upsilon. equation (Greek) character.
keyVval **OR** **keyvval**	Bl	Set vertical spacing (row height). The **v** or **V** keyletter parameter sets the vertical space for a table row corresponding to the **key** entry. It may be absolute, or (for **troff**) a signed digit increment or decrement value of the current vertical space (see topic 2.10.2.3).
x vec	Qk	Places a ( ) vector symbol above character **x** (see topic 2.9.10.4).
keyW(val) **OR** **keyw(val)**	Bl	Set minimum column width. The **(val)** is the minimum column width used; **val** may be scaled with **troff**, but the parentheses are optional if **val** is unscaled. The **val** is used for line length of text blocks in the affected column, but a column will be as wide as a data entry if the entry is wider than **val** (see topic 2.10.2.3).
XI	Qc	&Xi. equation (Greek) character.
xi	Qc	&xi. equation (Greek) character.
keyZ **OR** **keyz**	Bl	Zero-width item. The **z** or **Z** parameter to a keyletter causes **tbl** to ignore the corresponding data item width when calculating the column width to use (see topic 2.10.2.3).
zeta	Qc	&zeta. equation (Greek) character.
^	Bl	Vertically spanned row heading (see topic 2.10.2.3).
_	BlBd	As a column entry, the underscore indicates a horizontal line of column width (see topic 2.10.2.4). As table data, indicates a horizontal line the width of the table.
=	BlBd	As a column entry, indicates a double horizontal line of column width. As table data, indicates a double horizontal line the width of the table (see topic

		2.10.2.3).
|	Bl	Vertical bar. Used between column keyletters, it causes a vertical line to separate the columns the length of the table.
||	Bl	Double vertical bar. Used between column keyletters, cause a double vertical line to the separate columns the length of the table (see topic 2.10.2.3).
_	Bd	A short horizontal line data entry (see topic 2.10.2.4).
\^	Bd	Use the table entry directly above this data entry character and span it over this row, much like the table format keyletter **^** (see topic 2.10.2.4).
keyval	Bl	Column separation. **val** is a number value and scale indicator used as a space multiplier. The **val** (usually in ens) gives a column separation factor.
(~)	Qk	Forced blank character (see topic 2.9.10.1).
{ elm ...	Qk	Begin equation group (see topic 2.9.16).
elm ... }	Qk	End equation group (see topic 2.9.16).
"elm"	Qk	Delimiter within equations. Things within quote marks are printed literally and not interpreted or adjusted.
>=	Qc	**=** equation symbol.
<=	Qc	**=** equation symbol.
==	Qc	**==** equation symbol.
!=	Qc	**&nesym.** equation symbol.
+-	Qc	**±** equation symbol.
->	Qc	**→** equation symbol.
<-	Qc	**←** equation symbol.
<<	Qc	<< equation symbol.
>>	Qc	>> equation symbol.
(^)	Qk	Forced half-space blank character (see topic 2.9.10.1).
...	Qc	... treated as an equation element.
,...,	Qc	,..., treated as an equation element.

C.0 Appendix C. Phototypesetter Character Set

The following **troff** characters print in Roman, italic, and bold; they are entered as themselves:

```
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
a b c d e f g h i j k l m n o p q r s t u v w x y z
1 2 3 4 5 6 7 8 9 0 ! $ % & ( ) = | [ ] + ; : , . / ?
```

The following characters also print in Roman, italic, and bold; to get the one on the left, type the one-, two-, or four-character name on the right. (The ' symbol is the acute accent or apostrophe on most keyboards, while ` symbol is the grave accent; - is the minus sign on the keyboard.)

Sym.	**Call**	**Sym.**	**Call**	**Sym.**	**Call**	**Sym.**	**Call.**
ff	\(ff	fi	\(fi	fl	\(fl	ffi	\(Fi
ffl	\(Fl	_	\(ru	¼	\(14	¦	\(34
1/2	\(12	--	\(em	¦	\(co	°	\(de
&dagger.	\(dg	-	\-	¢	\(ct		\(rg
	\(bu	-	-	'	`	'	'
'	\(fm	&ballot	\(sq				

The following characters appear only on the special font:

Sym.	Call	Sym.	Call	Sym.	Call	Sym.	Call
\	\(e	-	\(rn	′	\('	`	\(`
+	\(pl	—	\(mi	×	\(mu	÷	\(di
=	\(eq	≡	\(==	≥	\(>=	≤	\(<=
≠	\(!=	±	\(+-	¬	\(no	/	\(sl
~	\(ap	′	\(~=	∝	\(pt	▽	\(gr
→	\(->	←	\(<-	↑	\(ua	↓	\(da
∫	\(is	∂	\(pd	∞	\(if	√	\(sr
⊂	\(sb	⊃	\(sp	∪	\(cu	∩	\(ca
⊆	\(ib	⊇	\(ip	∈	\(mo	φ	\(es
§	\(sc	‡	\(dd	⇐	\(lh	⇐	\(rh
⌈	\(lt	⌉	\(rt	⌈	\(lc	⌉	\(rc
⌊	\(lb	⌋	\(rb	⌊	\(lf	⌋	\(rf
⎨	\(lk	⎬	\(rk	|	\(bv	*	\(**
|	\(br	|	\(or	○	\(ci	ς	\(ts

The special font characters named **\(pl**, **\(mi**, *****, **\(sl**, and **\(eq**, which translate to +, -, *, /, and = respectively, are not the same as the current font characters named **+**, ****, **-**, *****, **/**, and **=**.

The following characters are also found only on the special font; they are entered as themselves:

```
# @ " { } < > ~ ^ _
```

The following pairs of input names are synonyms for each other:

\` \(ga

\' \(aa

- \(hy

Greek letters are also on the special font: all the lowercase letters including the terminal sigma (**\(ts**) and some uppercase letters (&gamma., &Delta., &Theta., &Lambda., &delta., &Xi., E, &Upsilon., &Phi., &Psi., and &Omega.) are true Greek characters. The remaining uppercase Greek letters are simulated by using the corresponding uppercase Roman letters. You can precede the Roman letter by **\(*** to get the corresponding Greek letter, for example, **\(*A** prints as A).

a b g d e z y h i k l m n c o p r s t u f x q w
&alpha. ß &gamma. &epsilon. &zeta. &eta. &theta. &iota. &kappa. &lambda. ¦ &nu

A B G D E Z Y H I K L M N C O P R S T U F X Q W
A B &Gamma. &Delta. E Z H &Theta. I K &Lambda. M N &Xi. O &Pi. P &Sigma. T &Up

D.0 Appendix D. Error Messages

The following error messages may be generated by text formatting commands. You may prefer to remove this appendix from this book and place it at the back of *Messages Reference*.

Subtopics
D.1 mm Error Messages
D.2 Formatter Error Messages

D.1 mm Error Messages

An **mm** error message has a standard string followed by a variable string. The standard string has the form:

ERROR:(filename) message-string line-number:

The variable **message-string** consists of a descriptive message, usually beginning with a macro name. The variable parts are listed in alphabetical order of the message string, followed by an explanation.

Check TL, AU, AS, AE, MT sequence

Cause: Something disturbed the correct order of style macros at the start of a memorandum.

Action: See "Style Macro Sequence" in *Text Formatting Guide* for the correct order.

Check TL, AU, AS, AE, NS, NE, MT sequence

Cause: Something disturbed the correct order of style macros at the start of a memorandum. (You used the macro and parameter **.AS 2** to generate this message instead of the preceding message.)

Action: See "Style Macro Sequence" in *Text Formatting Guide* for the correct order.

CS:cover sheet too long

Cause: The text of the memorandum cover sheet was too long to fit on one page.

Action: Reduce the abstract in type size or quantity or decrease the indent of the abstract (see "Abstract and Memorandum for File Cover Sheet" in *Text Formatting Guide*).

DE:no DS or DF active

Cause: A **.DE** macro was encountered but a previous **.DS** or **.DF** macro to end was missing.

Action: Add a **.DE**, a **.DF**, or a **.DS** macro statement.

DF:illegal inside TL as AS

Cause: Displays are not allowed in the memorandum title or abstract (see "Displays" in *Text Formatting Guide*).

Action: Remove the display from the title or abstract.

DF:missing DE

Cause: A **.DF** occurred within a display; displays cannot be nested.

Action: Add a **.DE** macro or correct a typographical error.

DF:missing FE

Cause: A display started inside a footnote; footnotes and displays cannot be nested.

Action: Add an **.FE** macro to end a previous footnote or correct a typographical error (see note, "Displays" and "Delimiting Footnote Text" in *Text Formatting Guide*).

DF:too many displays

Cause: More than 26 floating displays were accumulated but not yet output in your memorandum. The queue limit is 26 floating displays. (Usually, it is inappropriate to float a long sequence of displays.)

Action: Intersperse text or use some static displays.

DS:illegal inside TL or AS

Cause: Displays are not allowed in the title or abstract.

Action: Do not nest displays with themselves, titles, footnotes, or abstracts.

DS:missing DE

Cause: A **.DS** macro occurred within a display.

Action: Add a **.DE** macro, correct a typographical error, or check if a display is nested within a display (see "Displays" in *Text Formatting Guide*).

DS:missing FE

Cause: A display started inside a footnote.

Action: Add a **.FE** to end a previous footnote or correct a typographical error (see "Displays" in *Text Formatting Guide*).

FE:no FS active

Cause: An **.FE** macro was encountered with no previous matching **.FS** ("Delimiting Footnote Text" in *Text Formatting Guide*).

Action: Add a **.FE** macro or correct a typographical error.

FS:missing DE

Cause: A footnote started inside a display.

Action: Add a **.DS** or **.DF** to match the **.DE** (see "Displays" in *Text Formatting Guide*).

FS:missing FE

Cause: An **.FE** was not found for a previous **.FS**.

Action: An attempt is probably being made to begin a footnote within another one (see "Delimiting Footnote Text" in *Text Formatting Guide*); make sure footnotes are not nested.

H:bad arg:num

Cause: A bad **num** was supplied as a parameter to a **.H** macro.

Action: You must give a single digit **num** as the first parameter to **.H**; **num** can be a digit from **1** to **7**.

H:missing arg

Cause: An **.H** macro does not have a parameter.

Action: You must give **.H** at least one parameter.

H:missing DE

Cause: A heading macro (**.H** or **.HU**) occurred inside a display.

Action: You must add a **.DE** macro or correct a typographic error.

H:missing Fe

Cause: A heading macro (**.H** or **.HU**) occurred inside a footnote.

Action: You must add an **.Fe** macro or correct a typographic error. (See "Delimiting Footnote Text" in *Text Formatting Guide*).

HU:missing arg

Cause: An **.HU** macro does not have a parameter.

Action: You must give **.HU** macros at least one parameter.

LB:missing arg(s)

Cause: An **.LB** macro has one or more missing parameters.

Action: Give **.LB** at least four parameters (see "List Begin and Customized Lists" in *Text Formatting Guide*).

LB:too many nested lists

Cause: Another list was started when there were already six active lists.

Action: Nest lists only to six levels (see "Lists" in *Text Formatting Guide*).

LE:mismatched

Cause: An **.LE** macro occurred without a matching previous **.LB** or list-initialization macro that called **.LB** (see "List End" in *Text Formatting Guide*).

Action: This is not necessarily an error; a message is issued because there almost certainly is some problem you should look for in the preceding text.

LI:no lists active

Cause: An **.LI** occurred without a preceding list-initialization macro (see "List Items" in *Text Formatting Guide*).

Action: Look for an omitted list-initialization macro or a list-initialization macro separated from the **.LI** by an intervening **.H** or **.HU** macro. The **.LI** may be a typographic error.

ML:missing arg

Cause: An **.ML** macro was used without any parameters.

Action: You must give the **.ML** macro at least one parameter (see "Marked List" in *Text Formatting Guide*).

ND:missing arg

Cause: An **.ND** macro was used without any parameters.

Action: You must give the **.ND** macro one parameter (see "Setting the Date" in *Text Formatting Guide*).

RF:no RS active

Cause: An **.RF** does not have a previous matching **.RS**.

Action: You must give a preceding **.RS** macro or correct a typographic error (see "Delimiting References" in *Text Formatting Guide*).

RP:missing RF

Cause: A closing **.RF** macro was not found for a previous **.RP** and **.RS**.

Action: You must give an **.RF** macro for an **.RP** macro (see "Reference Page" in *Text Formatting Guide*).

RS:missing RF

Cause: A previous **.RS** macro was not matched by a closing **.RF**.

Action: You must give an **.RF** macro for a preceding **.RS** (see "Delimiting References" in *Text Formatting Guide*).

S:bad arg:value

Cause: An incorrect parameter **value** was given for **.S**.

Action: You must give a **value** for an **.S** macro that is a valid **troff** point size and scale indicator (see "troff Point Size and Vertical Spacing" in *Text Formatting Guide*).

SA:bad arg:num

Cause: An incorrect parameter (**num**) is given for an **.SA** macro.

Action: You must provide either **0** or **1** as the parameter (see "Right Margin Justification" in *Text Formatting Guide*).

SG:missing DE

Cause: An **.SG** macro occurred inside a display.

Action: You must end the display with a **.DE** macro before the closing signature is reached (see "Displays" in *Text Formatting Guide*).

SG:missing FE

Cause: An **.SG** signature macro occurred inside a footnote.

Action: Add an **.FE** macro (see "Footnotes" in *Text Formatting Guide*).

SG:no authors

Cause: An **.SG** macro occurred without any previous **.AU** macro calls (see "Author(s)" and "Signature Block" in *Text Formatting Guide*).

Action: You must provide author information for the memorandum macros to prepare the signature line.

VL:missing arg

Cause: A parameter was not provided with a **.VL** macro.

Action: You must give a **.VL** macro at least one parameter (see "Variable-Item List" in *Text Formatting Guide*).

WC:unknown option

Cause: An incorrect parameter was given to a **.WC** macro.

Action: You must either change the incorrect parameter or add a correct parameter (see "Two-Column Format" in *Text Formatting Guide*).

D.2 Formatter Error Messages

Most messages issued by **nroff** and **troff** are self-explanatory. Those error messages over which you have some control are listed here. If you receive other (**troff** or **nroff**) messages, follow your local procedures for reporting software problems. You should report other error messages to your local source of system support.

Cannot do ev

Cause: This message is caused by one or more of these:

Setting a page width that is negative or extremely shor
Setting a page length that is negative or extremely shor
Reprocessing a macro package (for example, performing a **.so** formatter request with input, on a macro package that is already requested on the command line)
Requesting the **-s1** flag of **troff** on a document that is longer than 10 pages.

Action: Correct as appropriate.

Cannot execute filename

Cause: The **.!** request cannot find the file **filename**.

Action: Make sure you are requesting a valid file name.

Cannot open filename

Cause: The file **filename** in the list of files to be processed cannot be opened.

Action: Make sure you have the necessary permissions.

Exception word list full

Cause: There are too many characters to be stored in the hyphenation exception list (via **.hw** requests).

Action: The exception list is limited. Remove unused words from the exception list or use inline hyphenation indicators within the text to avoid overfilling the list.

Line overflow

Cause: The output line being generated is too long for the line buffer of the formatter. (The excess is discarded by truncation.) See the **Word overflow** message.

Action: Adjust the line length, margin(s), type size, or text layout as needed.

Non-existent font type

Cause: A request was made to mount an unknown font.

Action: Request a font that is known to the formatter and available on the physical device.

Non-existent macro file

Cause: A requested macro package or file of custom macros cannot be found.

Action: Make sure the requested file was called correctly. Use an available macro package.

Non-existent terminal type

Cause: The command line flag for a terminal option referred to an unknown terminal type.

Action: Specify a known terminal type. If your terminal is not a known terminal type, specify a type it emulates. The default terminal type (Teletype 37) is usable for most terminals, though all features of the terminal may not be supported.

Out of temp file space

Cause: The needed additional temporary space for macro definitions, diversions, and so forth, cannot be allocated.

Action: Check first for one of these conditions:

Unclosed diversions (missing **.Fe** or **.DE**)
Unclosed macro definitions (missing **..**)
A huge table of contents

Too many number registers

Cause: The pool of number register names is full.

Action: Remove unneeded registers using the **.rr** request.

Too many page numbers

Cause: The list of pages specified to the formatter with the **-o** flag is too long.

Action: Use a more brief list (with fewer exceptions) or print the entire document.

Too many string/macro names

Cause: The pool of token and macro names is full.

Action: Remove unneeded macros and tokens using the **.rm** request.

Word overflow

Cause: A word being generated exceeded the formatter's word buffer. The excess characters were discarded by truncation. Likely causes for this and for the **Line overflow** message are:

Very long lines or very long words caused by either of
- The misuse of **\c** escape request
- The misuse of **.cu** line request

Very long equations produced by **eqn** or **neqn**.

Action: Correct as appropriate. Usually there is a formatting error; the word buffer is large enough for normal purposes. Large equations can often be restated in a more compact form.

D

F

M

N

W