intel®

XENIX* 286
TEXT FORMATTING GUIDE

*XENIX is a trademark of Microsoft Corporation.

Order Number: 174541-001

XENIX* 286
TEXT FORMATTING GUIDE

Order Number: 174541-001

*XENIX is a trademark of Microsoft Corporation.

REV.	REVISION HISTORY	DATE
-001	Original issue	11/84

intel®

TABLE OF CONTENTS

CONTENTS

Introduction

XENIX offers powerful writing and formatting tools for people who need to produce documents. Since both documents and programs are text files, all users share text editors, and commands for checking files and manipulating their contents. Those users who need to produce documents with professional formats can add text formatting tools.

Text processing is the broad set of activities that includes creating text, editing text, and producing professional, formatted documents. This book concentrates on text formatting and special tools for improving literary quality. It also gives hints for using the general-purpose commands to advantage when you are producing documents.

This introduction describes:

- The concepts and terms you will need to understand

- The editing and formatting tools you will be using

- The steps in the process of creating a finished document

- The strategies for managing writing projects

As you read this book remember that XENIX has been evolving over a number of years and that it offers an enormous range of tools. These tools have been developed independently, by many different programmers, so their capabilities often overlap. A large part of learning to use your XENIX system successfully is deciding how to make the various programs and utilities work together.

This manual is designed to help you approach a wide range of editing and formatting tools gradually. There are many programs described here for which you may not have an immediate application, and some you may never need at all. You need not learn all the material introduced here to produce professional-quality manuscripts. Choose the tools that will work best for your projects.

Before You Begin

Before you can begin to use your XENIX system effectively as a text processing environment, you should already be familiar with the material covered in the *XENIX 286 User's Guide*, particularly:

- The most common XENIX commands

- The XENIX hierarchical file structure

- The XENIX shell programming language

- At least one of the XENIX text editors

Equally important, however, is making use of the power of XENIX as an operating system by using its features to your advantage. In particular, consider how your work can be made easier by utilizing the XENIX hierarchical file structure to organize files efficiently. Make use of the XENIX shell to pipe one process to another and run several processes concurrently. Use the XENIX shell programming language to create scripts for automating your work. Develop strategies for managing your writing projects beyond merely learning a collection of commands.

Most importantly, before you try to use text formatting tools, learn one of the XENIX text editors well enough to enter and revise text.

Because there is so much to learn, read through this volume and decide which editors, utilities, and formatters best suit your needs. Then learn those tools that are most appropriate. As you become more experienced, you will develop a feel for which functions work best in which situations, and you will find new ways to make the writing process more efficient.

Overview of This Manual

This manual contains the following chapters:

1. Overview
    The chapter you are now reading describes XENIX tools and how they fit into each phase of document production.

2. Tools for Writing and Editing
    This chapter introduces several XENIX programs which can help you search for recurring patterns, compare files, and make global revisions to large files and groups of files. It also introduces three special writing tools that locate spelling errors, point out awkward diction, and assess the readability of a document.

3. Using the **MM** Macros
    This chapter introduces **MM**, a package of document formatting requests which simplifies the task of formatting documents.

4. **MM** Reference
    This chapter is a comprehensive guide to **MM**.

5. Using **Nroff/Troff**
    This chapter introduces the two XENIX text formatters, **nroff** and **troff**.

6. **Nroff/Troff** Reference
    This chapter is a comprehensive guide to the **nroff** and **troff** formatting programs.

7. Formatting Tables
    This chapter describes the specialized formatter, **tbl**, which produces effective tables in documents.

8. Formatting Mathematics
    This chapter describes the **eqn** program, which formats mathematical symbols and equations.

Basic Concepts

This section reviews some general text processing terms and concepts, including the:

- Types of writing tasks which can be done with XENIX

- Parts of a document

- Design characteristics of a formatted document

- Types of XENIX tools which you will be using

Writing Tasks

You can write, edit, and typeset any manuscript on the XENIX system--whether a memo, business letter, novel, academic dissertation, feature article, or manual. This manual uses many examples relevant to technical documentation, because these projects require the application of the greatest number of XENIX tools, and demand careful planning and strategy in their construction.

Anatomy of a Document

To fully determine the scope of your formatting needs, let's look at the parts of a typical document. Unless you are using XENIX to write memos and letters, you may have some or all of the following in your documents:

Front Matter

- Title page

- Copyright notice or document number

- Table of contents

- List of tables or illustrations

- Foreword

- Preface

- Acknowledgements

Body of Text

- Chapters or sections

- Figures and display

- Tables and equations

- Footnotes

- Running headers and footers

Back Matter

- Appendices

- Notes

- Glossary

- Bibliography

- Index

Your XENIX tools can automatically generate many parts of your document. For example, as part of the formatting process, you will be able to create lists of figures and tables, and a table of contents. You can create and store in advance a standard copyright notice page (often called a "boilerplate") and change only that information specific to the document.

Even in those sections of your document that must be written from scratch you can do much to standardize the "look" of a preface page, the pagination of an appendix, or the section numbering and format of a chapter. Once you have developed specifications, you can achieve consistency in the production of a long and complex document, and even produce many documents with the same specifications, without going through the definition process again. A further advantage is that you can change your specifications at any time, often without re-editing the text and formatting commands themselves. Then, you need only reformat your document and print it.

Formatting Characteristics

There are many characteristics of your finished text that can be controlled with XENIX formatting tools. Keep in mind, however, that the appearance of your finished document depends largely on the capabilities of your output device. To determine the format of your text you will insert commands in your text file as you write and edit. These commands will be identical, whether you are planning to produce your document on a lineprinter using the XENIX formatter **nroff**, or whether you are sending your document directly to a phototypesetter using **troff**. Because a lineprinter cannot do variable spacing, or change the point size or font of your text, **nroff** will ignore commands to change point size, round the parameters of spacing commands to the nearest line unit, and replace italics with underlining.

You will also notice qualitative differences in the output. For example, the justification of text--the spacing of text across the line to preserve a margin--is considerably less subtle in lineprinter output. Some of the characteristics you can control with the **nroff/troff** programs are:

- Text filling, centering, and justification

- Multicolumn output, margin, and gutter width

- Vertical spacing, line length, page length, and indentation

- Font type and point size

- Style of page headers and footers

- Page and section numbering

- Layout of mathematical equations and tables

An Inventory of Tools

When you approach any writing project, you should examine the whole range of XENIX tools to find those that will work best, just as you might look inside a toolbox. Although you can often do a job in several ways, there is frequently a tool, or a combination of tools, designed especially for that job.

Feel free to experiment in using the various editors, utilities, and formatters. If you make copies of your files and back up your XENIX system regularly, you can do little irreversible damage. As you work, you will gain more confidence and find new solutions.

While it is a good idea to learn to use a few of the XENIX tools skillfully, you should also strive to learn new tools and methods, rather than depending on a few procedures which you feel you know well. Some XENIX tools, like the screen editor **vi**, offer many more commands and functions than you can comfortably learn at one sitting. You may find yourself relying quite heavily on a limited number of commands. To prevent this, periodically review the documentation and try new commands.

In this manual we will be looking at XENIX tools that fall into a few basic categories:

System features

    Aspects of the XENIX operating system that can be used to enhance the text processing environment, such as multitasking and the hierarchical file structure.

Utilities

    These include the XENIX text editors (such as **vi**) and other utilities that are used for both software development and text processing (such as **sort, diff, grep,** and **awk**).

Text Formatting Tools

> These include specialized programs designed solely for text formatting tasks, including **MM, eqn,** and **tbl** and the formatters **nroff** and **troff.**

Writing Tools

> Special writing tools include **spell, style,** and **diction.**

Formatting Documents

In this section you will be introduced to **nroff** and **troff,** the two XENIX formatting programs for the lineprinter and phototypesetter, respectively. By inserting a series of commands in your text files you will be able to produce text with justified right margins, automatic page numbering and titling, automatic hyphenation, and many other special features. **Nroff** (pronounced "en-roff") is designed to produce output on terminals and lineprinters. **Troff** (pronounced "tee-roff") uses identical commands to drive a phototypesetter. The two programs are completely compatible, but **troff** output can be made considerably more sophisticated than output from **nroff.** With **troff,** for example, you can specify italic font, variable spacing, and point size. If you format the text using the same macros with **nroff,** italicized text will be underlined, the spacing will be approximated, and the text will be printed in whatever size of type the lineprinter offers.

The MM Macros

To use **nroff** and **troff,** you must insert a series of commands directly into your text. These formatting commands specify in detail how the final output will look. Because **nroff** and **troff** are relatively hard to learn to use effectively, XENIX also offers a package of formatting requests called the **MM** macros. With **MM** you can specify the style of paragraphs, titles, footnotes, multicolumn output, lists and so on, with less effort and without learning **nroff** and **troff** themselves. The **MM** program reads the commands from the text, and translates them into **nroff/troff** specifications. **MM** is described in detail in the next two chapters. It is recommended that you learn **MM** first, and use it for most of your formatting needs. If you need to fine-tune your output, you can add **nroff/troff** requests to the text as necessary.

To produce a document with **MM,** use the command

        nroff -mm filename

to view the output on your terminal screen. To store the output of **nroff** in a file, use the command line

        nroff -mm filename>outfile

where *outfile* is the name of the file you wish to designate for the stored output. It is suggested that you give consistent extensions to your input and output filenames. You might use ".s" for "source" as the extension for all input filenames, and ".mm" as the extension for the names of files which are the output of **MM**. For example,

        nroff -mm 1.intro.s>intro.mm&

Note: The ampersand tells the computer to process the file in the background. This enables you to continue working while the file is being processed.

Supporting Tools

In addition to the **nroff** and **troff** formatting programs, and the **MM** formatting package, there are other formatting programs to meet specialized needs. The **eqn** program, for example, formats complicated mathematical symbols and equations. A version of **eqn** called **neqn** outputs the same mathematical text for the more limited capabilities of lineprinter. **Eqn** is a preprocessor. You must run **eqn** first, before **nroff/troff**, to translate the commands of the **eqn** "language" into ordinary **nroff/troff** requests. The **eqn** commands resemble English words (e.g., over, lineup, bold, union), and the format is specified much as you might try to describe an equation in conversation. It is recommended that you delay learning about **eqn** in detail until you actually need to use it.

The **tbl** program is also a preprocessor: **tbl** commands are translated into **nroff/troff** commands to prepare complex tables. **Tbl** gives a you a high degree of control over material which must appear in tabular form, by doing all the computations necessary to align complicated columns with elements of varying widths. Like **eqn**, it requires that you learn another group of commands and process your files through another program before using **nroff/troff**.

Order of Invoking Programs

After you have inserted all your formatting commands into the text, you are ready to process your files, using the XENIX formatting programs. Please note that it is extremely important to use the various macro packages and formatters in the correct order. However, you may invoke all these programs with a single command line, using the XENIX pipe facility. As noted above, you can invoke the **MM** macro package along with **nroff/troff** using a command such as:

        nroff -mm intro.s>intro.mm

However, if you are using several specialized formatters along with **nroff/troff**, the command becomes more complex. You must invoke **eqn** before **nroff/troff** and **MM**, in order to translate the **eqn** commands into **nroff/troff** specifications before the files are

formatted, as in the following:

    neqn intro.s I nroff -mm > intro.mm

If you are using both **eqn** and **tbl,** the **tbl** program should be called first:

    tbl intro.s I neqnInroff -mm > intro.mm

If you are formatting multicolumn material or tables with **nroff** you must use the **col** (for "column") program. **Col** processes your text into the necessary columns, after formatting, as in:

    nroff -mm intro.s I col > intro.mm

A Sample Project

The preparation of every document has several phases: entering and editing text, checking your draft for spelling errors and style quality, formatting the finished version, and printing it on a printer or typesetter. To illustrate the process of producing a finished document with XENIX, let's look at the steps for creating a simple document.

Entering Text and Formatting Commands

First, you must write the text of the document. To do this, you will invoke one of the XENIX text editors and type the text on the screen. For example, to produce a memo informing the members of your department that you will be holding a seminar on XENIX text processing, you might begin by typing the following command line:

    vi memo.s

This calls the text editor **vi** and opens a file named "memo.s".

You will probably use your editor's special functions to correct errors and make revisions as you write, such as deleting words or lines, globally substituting one word for another, or moving whole paragraphs and sections around in the document.

If you have used a dedicated word processing system or a microcomputer word processing program before, note that XENIX formatting works somewhat differently. Instead of using special function keys to format your text on the screen as you work, you will be interspersing commands with ordinary text in your file.

If you are new to text formatting, we recommend you use **MM.** These macros replace whole sequences of **nroff** and **troff** commands and allow you to reduce the number and complexity of the commands necessary to format a document. You can use the **MM** macros wherever possible and add extra **nroff** or **troff** commands, as necessary, to fine-tune the format of your document.

Let's look at the beginning of a file called *memo.s*:
The **nroff/troff** command .ce (on the first line) indicates that the following text is to be centered. The **MM** macro .B stands for "boldface". In the final document, the word "MEMO" will appear centered on the page and in boldface.

```
.ce
.B MEMO
.sp 2
.P
A seminar has been scheduled for Thursday, September 15,
to introduce users to XENIX text processing.
It is is intended for all department members
planning to use XENIX for writing or preparing documentation.
.P
The seminar will include the following topics:
.AL 1
.LI
Reviewing the XENIX file structure and basic commands.
.LI
Using the vi text editor.
.LI
Formatting documents with mm.
.LE
.P
The seminar will begin at 9 A.M. and will last approximately
two hours...
```

The **nroff/troff** command .sp 2 means to space down two lines.

Each paragraph of text begins with the **MM** paragraph macro, .P.

The three **MM** macros .AL, .LI, and .LE turn the text following the words after "topics" into an automatically numbered list.

Note that all commands begin with a period in column one on the screen.   **MM** macros are capital letters, while **nroff/troff** commands are lowercase.

MM macros are discussed in detail in Chapters 3 and 4 of this manual.   Chapters 5 and 6 examine **nroff** and **troff** commands.

Formatting Text

Now, let's format the finished memo and place it in the file called *memo.s*. Use the following command line:

    nroff -mm memo.s>memo.mm&

This command invokes the **nroff** formatter using the **MM** macro package to format the file *memo.s*. When formatted, the memo will be stored in an output file called *memo.mm*. If you do not specify an output file, the formatted text will simply roll across your screen and be lost. Note that the command line ends with an ampersand (&), an instruction to put the formatting of this file "in the background". It is a good idea to put formatting jobs in the background because they will often take several minutes, especially if the file is long and the formatting complex. If you put the formatting job in the background, your terminal will remain free for you to do other work on the system.

Printing the Document

When you are ready to print the memo, use the command

    lpr memo.mm

to send the formatted file to the lineprinter.

The finished memo looks like this:

MEMO

A seminar has been scheduled for Thursday, September 15,
to introduce users to XENIX text processing.
It is intended for all department members planning
to use XENIX for writing or preparing documentation.

The seminar will include the following topics:

1. Reviewing the XENIX file structure and
basic commands.

2. Using the vi text editor.

3. Formatting documents with mm.

The seminar will begin at 9 A.M and will last
approximately two hours...

Managing Writing Projects

Once you have mastered one or more of your text editors and you are ready to do
extensive writing, revision, and text processing with the XENIX system, it is time to
consider the overall organization of your writing projects. This section offers some
common-sense suggestions for managing and standardizing your text files to make
processing more efficient. Not all of the suggestions and writing aids discussed here
will be equally appropriate in all situations. The larger and more complex the writing
project, however, the more time and confusion can be saved by the use of these
guidelines.

The Life Cycle of a Document

Before you can begin to work successfully with XENIX text processing tools, you need to
determine which tools are appropriate for each phase of a project. This section
discusses the application of XENIX tools to each step in the life cycle of a document
from the first outlines you develop to the archiving and management of multiple
versions and updates.

Every document goes through several phases before it is complete. First, you must type
the body of the text, using one of the XENIX text editors. As you write, you will insert
formatting commands, or macros, which specify how the final output should look.

In addition to checking your work for mistakes and spelling errors, you may need to go through an extensive revision process--the global substitution of one name or term for another, for instance, or the reorganization of your manuscript using a cut and paste technique.

Depending on the size and scope of your project, you may need to compare text variants and maintain several versions of your documents. Finally, you will be producing formatted output, whether it is a one-page business letter produced on a lineprinter or a book-length manuscript communicated directly to a phototypesetter. XENIX provides all the necessary tools for every phase of document preparation, and in many cases offers several approaches to each task.

Organizing Your Project

Organization is a key element of writing projects, especially if you are working on a large document or attempting to control many short ones. Text processing can greatly simplify any writing project if you use common sense in adapting the wide range of XENIX tools to your work. The time you spend defining the scope of your project in advance will be well rewarded. Decide which files and versions you need to maintain, and which formatting and error-checking programs you need to use. Determine in advance, if possible, the style and format of your text.

Most documents go through several revisions before they are finished. A few simple measures can make the work of repeated revision considerably easier. You will probably want to rewrite phrases and add, delete, or rearrange sentences. Subsequent editing of your text will be easier if every sentence starts on a new line, and if each line is short and breaks at a natural place, such as after a semicolon or comma.

As you are editing, you can insert markers in your text, so that you can return to them later. Use an unlikely pattern as a marker that you can search for easily using the **grep** command or your text editor. If, for example, you are unsure of which term to use, or how you want the final text to look, use a given word, or text formatting macro consistently. In this way, a global substitution can be made easily.

Certain global definitions, like the choice of a font for a given header level, or a commonly used string, may be created at the last minute and placed at the beginning of your text file. When you are experienced in the use of macros, you may want to create definitions which you use repeatedly. You can place your definitions in a separate file to be called every time you process your documents. This will facilitate consistency in your documents and allow greater flexibility if changes are required. In many cases, you will find that you can delay your formatting decisions until the document is ready to be printed or typeset.

Long documents should be broken down into individual files of reasonable length, perhaps ten to fifteen thousand characters. Operations on larger files are considerably slower, and the accidental loss of a small file is less catastrophic. If possible, each file should represent a natural boundary in a document, such as a chapter or section. Develop naming conventions to make your filenames consistent and self-explanatory, such as:

    1.intro.s   2.basic.s   3.adv.s

This allows files to be processed in groups with global commands, editing and shell scripts. You will also be able to see the contents of files and directories at a glance, and if someone else needs to access your files, they will not be confronted with files named "aardvark", "katmandu", or "fred".

You should also use the XENIX hierarchical file structure to your advantage in organizing your work, by creating different directories for special purposes. For example, you may wish to have your source text files in a different directory from your formatted output files, or you may find it handy to have "rough" and "final" draft directories. If your projects grow and change over time, you may need to maintain several versions of a document at once.

If you have created definition files and scripts, such as shell programs for processing text or **sed** scripts for making uniform changes, place them in yet another directory. This might also be a good place to add some "help" files to explain which versions of a document are contained in the directory or explain formatting procedures.

There are no rules to apply in deciding which procedures will produce documentation with the least effort and the fewest errors. How elaborate you make your procedures depends on the quantity and complexity of the text you need to process and maintain. Select the XENIX tools which seem most appropriate and adapt them to your own specific needs. The more organized and consistent your work is, the more powerful your use of these tools will become.

Shortcuts: The Boilerplates Concept

You will frequently find several approaches to any writing or revision with the XENIX system. Begin each writing project by reviewing these alternatives, and determine which solution requires the least repetitive effort and leaves the least room for error. You can increase your productivity, whether you are writing technical papers, documentation, or many memos with similar content, by writing clearly and concisely, rather than wasting time on needless duplication of effort. If you proceed in an organized, consistent way, as outlined in the previous section, you will find that XENIX offers you many shortcuts. One of these is the concept of the "editing script". The line editor **ed** can be used to perform a sequence of editing operations on a large group of files simultaneously. This can substitute for the use of a batch editing facility like **sed**, or **awk.**

For example, to change every "Xenix" to "XENIX" in all your files, create an **ed** script file with the following lines:

```
g/Xenix/s//XENIX/g
w
q
```

Now, you can use the command

```
ed filename <script
```

to make this change to any given file. The editor will take its commands from the prepared script. You can further automate procedures by using the XENIX shell language to write a shell procedure. For example, you can write a script which asks XENIX to make the above changes, reformat the entire text, and print the results. It is even possible to put this procedure in a file to be read by the "at" command to do your processing at some other time.

If you must produce many similar documents, or long documents which contain repeated material, the concept of the boilerplate may already be familiar to you. Often, information which must be presented in a standardized way can be stored in a separate file which can be used as necessary. This may be the preferred approach if a complex display or an example of program text must be reproduced. Using boilerplates assures consistency and makes subsequent changes to all recurrences of the copied material much simpler.

Summary

Here are some hints:

- Make your filenames easy to understand, and use a naming convention that allows you to take advantage of wildcard characters, the XENIX special characters. For an explanation of these special characters, see the *XENIX 286 User's Guide*.

- Create text files of manageable length which represent chapters or logical divisions in the document; arrange files into directories which represent major documents or versions so that they can be easily identified; move formatted output into directories separate from your source text.

- Create "help" or "README" files in each directory which explain your text--what version you are writing, what scripts, processors, and files are needed to successfully produce the document. Use comment lines in your text to explain organizational details of your project or any special macros you have created.

- Control parallel versions and updates carefully, especially if you are working on a large project. Use conditional processing in your text files, keep copies of text in different directories, and link files where appropriate. If you are in doubt about versions of text in different files use **diff** to compare text. See Chapter 2 to find out how to use **diff.**

- When using **vi** or another text editor to write text, start each sentence or clause on a new line.

- Identify text and formats which recur in a document or several documents, and create boilerplates or templates to save work.

- Make full use of cut and paste techniques to rearrange material in a file, move text between files, or use the same text repeatedly in several places. Cut and paste is explained in Chapter 2 of this manual.

- Use batch processes like **sed, awk,** or an **ed** script to make consistent changes to a large number of files. See the *XENIX 286 User's Guide.*

- Use **spell, style,** and **diction** regularly to reduce the number of editorial corrections.

- Try to define your production specifications and style conventions in advance; prepare editing scripts to reduce the number of changes you need to make individually.

- Always use the simplest possible technique to achieve your results. Use the **MM** macros where possible, reserving **nroff/troff** commands for "fine-tuning" or creating an effect impossible with **MM.** If you define a new macro, explain it in a comment line so it can be readily understood.

- Avoid running too many formatting processes simultaneously. If necessary, use the **at** command to process files at a time when the system is not busy.

- Protect yourself by backing up your system and user files regularly. Make copies of files if you are in doubt about whether your procedures will damage them.

Introduction

This chapter introduces you to some XENIX system utilities that can simplify document editing and revision. It also discusses three special XENIX writing tools for improving writing style and locating typographical errors in documents.

This chapter focuses on how the XENIX tools are used to accomplish some common text processing tasks. These tools are XENIX utilities which are also used by programmers for searching and editing data and program text. The emphasis here is on XENIX commands and utilities that can help you simplify complicated editing procedures, and allow you to work with many files at once. As you read, it will become apparent that several of the programs introduced here can be used interchangeably, and that many of these tasks can also be performed with your text editor. You may also find the two XENIX programs, **sed** and **awk**, helpful for making complex changes to text files.

There are several revision tasks common to all text processing projects. The larger your project, the more complex these tasks become. For example, you may need to change a key term, name, or phrase everywhere it appears, or locate references to items you need to change or delete. You may need to compare and contrast multiple versions of your text in order to locate variations. You may also need to alter some aspect of the text format to suit production requirements. To do any of these tasks, you must locate a string--a word, a phrase, a text formatting macro or any repeated set of characters-- and, if necessary, change it everywhere it appears. Using the XENIX system tools discussed in this chapter, these changes can be made rapidly and consistently.

The first half of this chapter discusses several easy ways to learn XENIX commands. If you have read the *XENIX 286 User's Guide*, you may already be familiar with some of them. More detailed information about these commands is provided in the *XENIX 286 Reference Manual*. The commands include:

- **Grep** commands print lines that match a single specified pattern. When combined with other commands in a shell procedure and used to process many files at once, the **grep** commands become extremely powerful for locating text in large files. Two variants of **grep** are also introduced in this chapter: **egrep** and **fgrep**.

- The XENIX file comparison utilities, **diff, diff3,** and **comm.** These utilities compare two or more files and output those lines which do not match. In text processing applications these programs can be extremely useful for quickly locating variations between several versions of documents.

- Additional XENIX commands, including **sort,** which alphabetizes lines in your text files; **wc,** which counts lines, words, and characters in your text; and **cut** and **paste,** which duplicates "cut and paste" editing operations.

XENIX Commands for Text Processing

Pattern Recognition: The Grep Commands

Because of its power to search for patterns in many files at once, **grep** and its variants are among the most useful XENIX commands. The members of the **grep** family, like the **awk** program and the batch editor, **sed**, have as their basis the same principle of pattern recognition as the text editors, **ed** and **vi**. Each of these programs searches for the occurrence of a given pattern--a character or group of characters, a word or word string--and generates a list of those lines containing the same pattern. Finding all occurrences of a word or pattern in a group of files is a common text processing task. You can easily write a shell script using the **grep** command or one of its variants, **egrep** and **fgrep**, and quickly search multiple files. **Grep** searches for the same regular expressions recognized by **ed**. The word "grep" stands for

g/re/p

that is, "globally" locate a pattern and then print it. **Grep** searches every line in a set of files for all occurrences of the specified regular expression. Thus,

grep thing file1 file2 file3

finds the pattern "thing" wherever it occurs in any of the files you name (e.g. *file1, file2, file3*). If you use the -n option with **grep**, it will indicate not only the file in which the line was found but also the line number, so that you can locate and edit it later. By combining the use of **grep** with other commands to generate a shell program that reads and transforms input, large quantities of text can be processed through multiple searching or editing procedures quickly.

The commands **grep, egrep,** and **fgrep** all search one or more files for a specified pattern. They appear on the command line in the following form:

grep [option] expression filename

Commands of the **grep** family search the files you specify (or the standard input if you do not specify any files) for lines matching a pattern. Each line is copied to the standard output (your terminal screen), but if you are processing great quantities of text you should specify a filename in which to store the results of the **grep** search.

For example, the command

    grep -n 'system utility' chap*.s >util

requests that **grep** command search for the phrase "system utility" in every file that begins with "chap" and ends with ".s", and store the resulting list, with line numbers, in a file called *util*. Unless the -h option is used, the filename is given if there is more than one input file.

The difference between the three **grep** variants is the type of expression you are allowed to search for. **Grep** searches for every regular expression and allows you to use the special characters to define special patterns. **Egrep** looks for the same regular expression as **grep,** but also has an extra set of characters that allows you to search for more than one occurrence of an expression, or more than one expression at a time. **Fgrep** can only look for strings; no special characters are allowed, and thus **fgrep** is faster than **grep** or **egrep.** For more information about **grep, egrep,** and **fgrep,** see **grep** in the *XENIX 286 Reference Manual.*

File Comparison: diff, diff3, and comm

In addition to locating occurrences of particular strings or regular expressions in your text, you will find it useful to compare and contrast two or more similar text files.

The **diff** command compares two files and outputs a list of differences. You can use **diff** to store file versions more compactly. This is accomplished by storing the output of **diff,** which would be the differences in that file version, rather than the file itself. The -e option collects a script of those **ed** commands (such as append, change, and delete) which would be necessary to recreate the revised file from the original.

Diff3 is similar to **diff,** but is used to compare three files.

Another comparison tool, **comm,** is discussed in this section. **Comm** is useful primarily for comparing the output of two sorted lists.

Diff

To use the **diff** command to compare two files, use the form:

    diff -option file1 file2

Diff reports which lines must be changed in two files to bring them into agreement. If you use a dash (-) instead of the first filename, **diff** will read from the "standard input". The normal output contains lines in this format, where *n* is the line number of the text file:

```
17a18
> line affected in file 2
23,25d 26
< line affected in file 1
< line affected in file 1
30c31
< line from file 1
---
> line from file 2
```

These lines resemble the **ed** commands which would be necessary to convert *file1* into *file2*. The letters *a*, *d*, and *c* are **ed** commands for appending, deleting, and changing, respectively. The line numbers after the letters refer to *file2*. Following each of these lines are printed all the lines that are affected in the first file, flagged by a less-than sign (<), then all the lines that are affected in the second file, flagged by a greater-than sign (>).

For example, you might want to compare two text files, *fruit* and *vegies*. The contents of the file called *fruit* are the lines:

```
apples
bananas
cherries
tomatoes
```

The contents of the file called *vegies* are the lines:

```
asparagus
beans
cauliflower
tomatoes
```

The command line

```
diff fruit vegies>diffile&
```

produces the file *diffile* that contains the list of differences between *fruit* and *vegies* which are the output of the **diff** program:

```
1,3c1,3
<apples
<bananas
<cherries
---
>asparagus
>beans
>cauliflower
```

In this case, lines 1 through 3 in the file *vegies* are different from lines 1 through 3 in the file *fruit*. See **diff** in the *XENIX 286 Reference Manual* for options.

Using Diff3

Diff3 works like **diff,** except that it compares three files. It has the form:

    diff3 -option *file1 file2 file3*

Diff3 reports disagreeing ranges of text flagged with the following codes:

    =====All three files differ

    =====1 File1 is different

    =====2 File2 is different

    =====3 File3 is different

The change which has occurred in converting a given range of lines in a given file to some other is reported.

For example, the message

    file1 : n1 a

means text is to be appended after line number n1 in file *file1.* The message:

    file1 : n1 , n2 c

means that the text to be changed is in the range of lines n1 to line n2. If n1 = n2, the range may be abbreviated to n1.

The original contents of the range follow immediately after a "c" indication. When the contents of two files are identical, the contents of the lower-numbered file is suppressed.

As in the case of **diff, diff3** used with the -e option prints a script for **ed** that will incorporate into *file1* all changes between *file2 file3.* In other words, it records the changes that normally would be flagged ==== and ====3.

Comm

The **comm** program selects or rejects lines common to two sorted files. It has the form:

    comm [-option] file1 file2

Comm reads *file1* and *file2*, and produces a three-column output: lines only in *file1*, lines only in *file2*, and lines in both files. Ordinarily, the lines should be sorted in ASCII collating sequence by using the **sort** program before using **comm**. As in **diff** and its variants, if you type a dash (-) instead of a filename, **comm** will read either *file1* or *file2* from the standard input.

The possible options with **comm** are the flags 1, 2, or 3, which suppress printing of the corresponding column. Thus **comm** with -12 suppresses printing of the first two columns and prints only the lines common to the two files; **comm** -23 prints only lines in the first file but not in the second. The command **comm** with the options -123 would print no lines.

Other Useful Commands

In this section a group of XENIX commands that are helpful in text manipulations are summarized. In each case you may find it helpful to refer to the *XENIX 286 Reference Manual* for more information.

Sort

If you have been using your XENIX system for a while, you may have already learned the **sort** command. Because of its capacity to alphabetize a list of items, it can be extremely useful in a variety of text processing situations (e.g., alphabetizing the names on a mailing list or the entries in an index). To use sort, simply type the command

    sort *file name* >list.out

The output file *list.out* will contain the sorted list.

Like some other XENIX commands, if you use "-" instead of a filename, **sort** will read from the standard input, and unless you direct the output to another file, the sorted list will appear on your screen. **Sort** will, by default, sort an entire line in ascending ASCII collating sequence, including letters, numbers, and special characters. See **sort** in the *XENIX 286 Reference Manual* for a list of available options.

If you need to do repeated sorts by field, you may find it easier to prepare a simple **awk** script.

Note that if you invoke one or more of the **sort** options, or use position names, you must use the following syntax:

    sort [*-options*] [*pos1*] [*pos2*] [*-o output*] [*filenames*]

Wc

The XENIX command **wc** counts words, characters, or lines in your files. If, for example, you are submitting a manuscript to a publisher, an exact word count may be necessary, or you may want to estimate the number of lines in your file before you make some critical formatting decision. To use **wc**, type

    wc filename

If you give no options, **wc** automatically counts lines, words, and characters in the named files, or in the standard input if you do not specify any filenames. It keeps a total count for all named files, and the filenames will also be printed along with the counts. The option -l for "lines," option -w for "words" and option -c for "characters" can be also be used in any combination, if you do not want all three statistics printed. Remember, when doing a word count, that **wc** will automatically treat as a word any string of characters delimited by spaces, tabs, or newlines.

Cut and Paste

If you work with large text files, you may find the two XENIX commands, **cut** and **paste**, extremely useful for rearranging text blocks within a document.

Cut is a useful shortcut for extracting columns or fields of information from a file, or for rearranging columns in lines. To invoke **cut** in its simplest form, type:

    cut [options] file

The **cut** command will cut out columns from each line of a file. The columns can be specified as fields separated by a named delimiter or by character positions. The following options are available:

-c*list*   A list of numbers following -c specifies character positions or ranges.

-f*list*   A list of numbers following -f is a list of fields, delimited by a character specified after the -d option.

-d*char*   A character following the -d option is read as the field delimiter. The default is the tab character. Spaces or other characters with special meanings must be surrounded with single quotation marks (').

-s   This option suppresses lines which do not contain the delimiter character, if the -f option is invoked.

Either the -c or -f option must be invoked when using **cut**.

The **paste** command performs the reverse operation: it can be used to merge lines in one or several files. To use **paste** in its simplest form, type

        paste file1 file2

Paste will concatenate *file1* and *file2*, treating each file as a column or columns of a table and pasting them together horizontally. As with the **cut** command, you can also specify a delimiter character to replace the default tab. You can even use **paste** to merge material in columns into lines in a single file.

The following options are available:

-d   The –d option suppresses the tab which automatically replaces the newline character in the old file. It can be followed by one or more characters which act as delimiters.

list   The list of characters which follow the –d option.

-s   The –s option merges subsequent lines, rather than one from each input file. The tab is the default character, unless a list is specified with the –d option.

-       The dash can be used in place of any filename, to read a line from the standard input.

There are, of course, several other ways to approach cut and paste operations with the XENIX system. By now you should feel fairly confident using one of the XENIX text editors to move blocks of text, write parts of files to new files, and rearrange lines. Using **sort** to alphabetically sort fields within lines, or the **awk** program to change the order of fields in a text file, are two special cases of cut and paste operations.

Writing Tools

In the previous sections you were introduced to some common XENIX utilities that can be used to search for patterns, do batch editing, or compare two or more files. This section introduces three XENIX programs which have been designed solely for writing and editing documents:

●   **spell,** a program that checks for spelling and typographical errors in your text files.

●   **style,** a program that analyzes the readability of your writing style, based on statistical measures of sentence length and type.

●   **diction,** a program that searches for awkward, ambiguous, and redundant phrases and suggests alternatives.

Think of these programs as tools. As you read about these programs, keep in mind that they are not intended to substitute for careful reviewing, editing, and proofreading on your part. Use **spell, style,** and **diction** early in the editing process as a preliminary check on your work. You will get some interesting feedback on your writing and uncover recurrent patterns in your word usage and sentence construction. Your common spelling errors will be pointed out. As you are preparing your final draft, you may wish to use **spell** again to locate any last-minute typographical errors.

Using Spell

You can save a lot of time and grief in proofreading your documents by using **spell.** Although not totally infallible, the **spell** program will find most of your spelling and typographical errors with a minimum of effort and processing time. **Spell** compares all the words in the text files you specify with the correctly spelled words in a pre-existing XENIX dictionary file. Words which neither appear in this dictionary, nor can be derived by the application of ordinary English prefixes, suffixes, or inflections are printed out as spelling errors. You can either specify an output file in which to store the list of misspelled words, or allow them to appear on your screen. For example, to find the spelling errors in a file named *1.intro.s,* type

       spell 1.intro.s

and a list of possible misspelled words will appear on your screen. You can also use a command line like

       spell *.s > errors&

to check all your files with names ending in ".s" at once and output the possible misspellings into a single file named *errors.*

Spell ignores the common formatting macros from **nroff, troff, tbl,** and **eqn.** It automatically invokes a program called **deroff** to remove all formatting commands from the text file being examined for spelling errors.

Several options are available. With "spell -v", words not literally in the dictionary are also printed, along with plausible derivations from dictionary words. The -b option checks British spelling. This option prefers British spelling variants such as: centre, colour, speciality, and travelled, and insists on the use of "-ise" in words like "standardise".

The XENIX dictionary is derived from many sources, and while it recognizes many proper names and popular technical terms, it does not include an extensive specialized vocabulary in biology, medicine, or chemistry. The XENIX dictionary will not recognize your friends' names, your company's acronyms, and many esoteric words, and will list them as spelling "errors". It is difficult to predict in advance which technical terms, names, and acronyms **spell** will uncover in your documents.

Using Style and Diction

This section describes two programs, **style** and **diction**. Although these two programs attempt to critique your writing style, keep in mind that the qualities which distinguish good writing from bad are not entirely quantifiable. Taste in writing remains subjective, and different stylistic qualities may be appropriate to different writing situations. XENIX is neither a literary critic nor your sophomore English teacher. These tools are best used to eliminate errors and give you some preliminary assessment of a document's readability. They are not intended to substitute for human editing.

Both **style** and **diction** are based on statistical measures of writing characteristics--characteristics that can be counted and summarized on your computer. With a large number of documents stored on computers it is has become feasible to study the recurrent features of writing style in a great many documents. The programs described here use the results of such studies to help you write in a more readable style. They produce a stylistic profile of writing, including:

- A measurement of readability, determined on the basis of sentence and word length, sentence type, word usage, and sentence openers.

- A listing of awkward, ambiguous, redundant and ungrammatical phrases found in the document.

This will help you evaluate overall document style, and correct or eliminate poor word choices or awkward sentences. As you work with these programs, you can accumulate data to provide you with a profile of your writing style based on all your documents.

Because the **style** and **diction** programs can only produce a statistical evaluation of words and sentences, the term "style" is defined here in a rather narrow way: the results of a writer's particular word and sentence choices. Although many stylistic judgements are subjective, particularly those involving word choice, these programs make use of some relatively objective measures developed by experts.

Three programs have been written to measure some of the objectively definable characteristics of writing style and to identify some commonly misused or unnecessary phrases. Although a document that conforms to these stylistic rules is not guaranteed to be coherent and readable, one that violates all of the rules will almost certainly be difficult or tedious to read. These programs are:

- **Style**, which calculates readability, sentence length variability, sentence type, word usage and sentence openers. It assumes that the sentences are well-formed, i.e., that each sentence has a verb and that the subject and verb agree in number.

- **Diction**, which identifies phrases that reflect dubious usage or seem unnecessarily awkward.

These programs are described in detail in the following sections.

Style

Style reads a document and prints a summary of sentence length and type, word usage, sentence openers and "readability indices." The readability indices are tradition school grade levels assigned to a document, based on four different studies of what makes one style more readable than another. You can also use the **style** program to locate all sentences in a document longer than a given length, those containing passive verb forms, those beginning with expletives, or those with readability indices higher than a specified number.

Style is based on a system called "parts", which determines parts of speech in the English language. **Parts** is a set of programs which uses a small dictionary and experimentally derived rules of word order to assign word classes to all words in the text. It can be used for any text with an accuracy rate of approximately 95%. Style measures have been built into the output phase of the programs that make up **parts.**

The style program is invoked with the following syntax:

    Style [options] file

What Is a Sentence?

A human reader has little trouble deciding where a sentence begins and ends. Computers, however, are confused by different uses of the period character (.) in constructions like 1.25, A. J. Jones, Ph.d., i.e., or etc. Before attempting to count the words in a sentence, the text is stripped of potentially misleading formatting macros. Then **style** defines a sentence as a string of words ending in one of the punctuation marks:

The end marker "/." may be used to indicate an imperative sentence. Imperative sentences not marked in this way are not identified. **Style** recognizes numbers with embedded decimal points and commas, strings of letters and numbers with embedded decimal points used in computer filenames, and a list of commonly used abbreviations. Numbers that end sentences cause a sentence break if the next word begins with a capital letter. Initials followed by periods are only assumed to be at the end of the sentence if the next word begins with a capital and is found in the dictionary of function words used by **parts.** As a result, the periods in the string

    J. D. Jones

are not read as the ends of sentences, but the period after the *H* in the following string is assumed to end a sentence:

    ...system H. The...

Using these rules, most sentences are correctly identified, although occasionally two sentences are counted as one or a fragment is identified as a sentence.

The results of running **style** are reported in five parts. A typical output might have values that look like this:

readability grades
      (Kincaid) 12.3 (auto) 12.8 (Coleman-Liau) 11.8 (Flesch) 13.5 (46.3)

sentence info
      no. sent 335 no. wds 7419 av sent leng 22.1 av word leng 4.91 no. questions 0 no. imperatives 0 no. nonfunc wds 4362 58.8%   av leng 6.38 short sent (<17) 35% (118) long sent (>32) 16% (55) longest sent 82 wds at sent 174; shortest sent 1 wds at sent 117

sentence types
      simple 34% (114) complex 32% (108) compound 12% (41) compound-complex 21% (72)

word usage
      verb types as % of total verbs tobe 45% (373) aux 16% (133) inf 14% (114) passives as % of non-inf verbs 20% (144) types as % of total prep 10.8% (804) conj 3.5% (262) adv 4.8% (354) noun 26.7% (1983) adj 18.7% (1388) pron 5.3% (393) nominalizations 2 % (155)

sentence beginnings
      subject opener: noun (63) pron (43) pos (0) adj (58) art (62) tot 67% prep 12% (39) adv 9% (31) verb 0% (1) sub_conj 6% (20) conj 1% (5) expletives 4% (13)

Readability Grades

The **style** program uses four separate readability indices. Generally, a readability index is used to estimate the grade level of the reading skills needed by the reader to understand a document. The readability indices reported by **style** are based on measures of sentence and word lengths. Although the indices themselves do not measure whether the document is coherent and well organized, high indices correlate with stylistic difficulty. Documents with short sentences and short words have low scores; those with long sentences and many polysyllabic words have high scores. Four sets of results computed by four commonly used readability formulae are reported: the Kincaid Formula, the Automated Readability Index, the Coleman-Liau Formula, and a version of the Flesch Reading Ease Score. Because each of these indices was experimentally derived from different text and subject results, the results may vary. They are summarized here.

Kincaid Formula

The formula is: Reading Grade=11.8 * syllables per word + .39 * words per sentence - 15.59

The Kincaid formula is based on Navy training manuals ranging in difficulty from 5.5 to 16.3 in grade level. The score reported by this formula tends to be in the mid-range of the four scores. Because it is based on adult training manuals rather than schoolbook text, this formula is probably the best one to apply to technical documents.

Automated Readability Index (ARI)

The formula is: Reading Grade=4.71 *letters per word + .5 * words per sentence - 21.43

The Automated Readability Index is based on text from grades 0 to 7, and intended for easy automation. ARI tends to produce scores that are higher than Kincaid and Coleman-Liau but are usually slightly lower than Flesch.

If you invoke **style** with the -r option followed by a number, all sentences with an Automated Readability Index equal to or greater than the number specified will be printed.

Coleman-Liau Formula

The formula is: Reading Grade=5.89 * letters per word - .3 * sentences per 100 words - 15.8

This is based on text ranging in difficulty from .4 to 16.3. This formula usually yields the lowest grade when applied to technical documents.

Flesch Reading Ease Score

The formula is: Reading Score=206.835 - 84.6 * syllables per word - 1.015 * words per sentence.

This formula is based on grade school text covering grades 3 to 12. The first number reported is the grade level of the document. The second number, in parentheses, is the difficulty score. It is usually reported in the range 0 (very difficult) to 100 (very easy).

The score reported by **style** is scaled to be comparable to the other formulas, except that the maximum grade level reported is 17. On the whole, the Kincaid formula is the best predictor for technical documents. Both ARI and Flesch tend to overestimate text difficulty; Coleman-Liau tends to underestimate. On text in the range of grades 7 to 9 the four formulas tend to be about the same. For easy text, use the Coleman-Liau formula since it is reasonably accurate at the lower grades.

It is generally safer to present text that is too easy than too hard. If a document has particularly difficult technical content, especially if it includes a lot of mathematics, it is probably best to make the text very easy to read. You can lower the readability index by shortening sentences and words, so that the reader can easily concentrate on the technical content.

Remember that these indices produce only rough estimates; the results should not be taken as absolute.

Sentence Length and Structure

The output sections labeled "sentence info" and "sentence types" give both length and structure measures. **Style** reports on the number and average length of both sentences and words. It also reports the number of questions and imperative sentences. "Nonfunction words" refer to all the nouns, adjectives, adverbs, and nonauxiliary verbs. Function words are prepositions, conjunctions, articles, and auxiliary verbs.

Since most function words are short, they tend to lower the average word length. The average length of nonfunction words, therefore, is a more useful measure for comparing word choice of different writers than the total average word length. The percentages of short and long sentences measure sentence length variability. Short sentences are those at least five words less than the average. Long sentences are those at least ten words longer than the average. Finally, the length and location of the longest and shortest sentences is reported in the "sentence information" section. If the flag – l*number* is used, **style** will print all sentences longer than the specified number.

Style applies the following rules to the definition of sentence types:

1.   A simple sentence has one verb and no dependent clause.

2.   A complex sentence has one independent clause and one dependent clause, each with one verb. Complex sentences are found by identifying sentences that contain either a subordinate conjunction or a clause beginning with a word like "that" or "who". The preceding sentence has such a clause.

3.   A compound sentence has more than one verb and no dependent clause. Sentences joined by a semicolon (;) are also counted as compound.

4.   A compound-complex sentence has either several dependent clauses or one dependent clause and a compound verb in either the dependent or independent clause.

Most authorities on effective writing style emphasize variety in sentence length, as well as overall sentence structure. Three simple rules for writing sentences are:

1.   Avoid the overuse of short simple sentences.

2.   Avoid the overuse of long compound sentences.

3.   Use various sentence structures to avoid monotony and increase effectiveness.

Word Usage

The word usage measurements used by **style** attempt to identify other features of writing constructions. In English, there are many alternative ways to say the same thing. For example, the following sentences all convey approximately the same meaning but differ in word usage:

•   The cxio program is used to perform all communication between the systems.

•   The cxio program performs all communications between the systems.

•   The cxio program is used to communicate between the systems.

•   The cxio program communicates between the systems.

•   All communication between the systems is performed by the cxio program.

The distribution of the parts of speech and verb constructions in a document helps the writer identify the overuse of particular construction. For each category, **style** reports a percentage and a raw count of the parts of speech used. Although these measures are somewhat crude, they demonstrate excessive repetition of sentence constructions. In addition to looking at percentages, it is useful to compare the raw count with the number of sentences. If, for example, the number of infinitives is almost equal to the number of sentences, then an unusual number of sentences in the document must contain infinitives, like the first and third sentences in the example above. You may want to change some of these sentences for greater variety.

Verbs

To determine the predominant verb constructions in a document, verb frequency is measured in several ways. Technical writing, for example, tends toward passive verb constructions and other usages of the verb "to be". The category of verbs labeled "tobe" measures both passives and sentences of the form:

subject tobe predicate

Whole verb phrases are counted as a single verb. Verb phrases containing auxiliary verbs are counted in an "aux" category, including verb phrases whose tense is not simple present or simple past. Infinitives are listed as "inf." The percentages reported for these three categories are based on the total number of verb phrases found. These categories are not mutually exclusive; some constructions may be in more than one category. For example, "to be going" counts as both "tobe" and "inf". Use of these three types of verb constructions varies significantly among different writers.

Style reports passive verbs as a percentage of the finite verbs in the document. Because sentences with active verbs are easier to comprehend than those with passive verbs, you should avoid the overuse of passive verbs. Although the inverted object-subject order of the passive voice seems to emphasize the object, studies show that comprehension is not significantly affected by word position. Furthermore, a reader will retain the direct object of an active verb better than the subject of a passive verb. The -p option causes **style** to print all sentences containing passive verbs.

Conjunctions

Conjunctions provide logical parallelism between ideas by connecting two or more equal units. These units may be whole sentences, verb phrases, nouns, adjectives, or prepositional phrases. The compound and compound-complex sentences reported under sentence type are parallel structures. Other uses of parallel structures are indicated by the degree that the number of conjunctions reported under word usage exceeds the compound sentence measures.

Adverbs

Adverbs provide transitions between sentences and order in time and space. Like pronouns, adverbs provide connectivity and cohesiveness.

Nouns and Adjectives

Some writers qualify almost every noun with one or more adjectives. If the ratio of nouns to adjectives in your text approaches one, it is probable that you are using too many adjectives. Multiple qualifiers in phrases like "simple linear single-link network model" often lend more obscurity than precision to a text.

Pronouns

Pronouns can add cohesiveness to a document by acting as a shorthand notation for something previously mentioned. Documents with no pronouns tend to be verbose and to have little connectivity.

Nominalizations

Nominalizations are verbs transformed into nouns by the addition of a suffix like: "ment", "ance", "ence", or "ion". Examples are accomplishment, admittance, adherence, and abbreviation. When a writer transforms a nominalized sentence to a non-nominalized sentence, it becomes more effective. The noun becomes an active verb and frequently one complicated clause becomes two shorter clauses. For example

Their inclusion of this provision is admission of the importance of the system.

could be changed to:

When they included this provision, they admitted the ...

The transformed sentences are easier to comprehend, even if they are slightly longer, provided the transformation breaks one clause into two. If your document contains many nominalizations, you may want to transform some of the sentences to use active verbs.

Sentence Openers

Another principle of style is the desirability of varied sentence openers. Because **style** determines the type of sentence opener by looking at the part of speech of the first word in the sentence, the sentences counted under the heading "subject opener" may not all really begin with the subject. However, a large total percentage in this category suggests a lack of variety in sentence openers. Other sentence opener measurements help determine if there are transitions between sentences and where subordination occurs. Adverbs and conjunctions at the beginning of sentences are mechanisms for the transition between sentences. A pronoun at the beginning of a sentence shows a link to something previously mentioned and indicates connectivity.

The location of subordination can be determined by comparing the number of sentences that begin with a subordinate conjunction with the number of sentences with complex clauses. If few sentences start with subordinate conjunctions then the subordination is embedded or at the end of the complex sentences. For greater variety, transform some sentences so that they have leading subordination.

The last category of openers, expletives, is commonly overworked in technical writing. Expletives are the words "it" and "there", generally used with the verb "to be" in constructions where the subject follows the verb. For example,

      There are three streets used by the traffic.
      There are too many users on this system.

This construction tends to emphasize the object rather than the subject of the sentence. The -e option will cause **style** to print all sentences that begin with an expletive.

Diction

The **diction** program prints all sentences in a document containing phrases that are either frequently misused or indicate wordiness. **Diction** uses **fgrep** to match a file of phrases or patterns to a file containing the text of the document to be searched. A data base of about 450 phrases has been compiled as a default pattern file for **diction.** To facilitate the matching process, **diction** changes uppercase letters to lowercase and substitutes blanks for punctuation before beginning the search for matching patterns. Since sentence boundaries are less critical in **diction** than in **style,** abbreviations and other uses of the period character (.) are not treated specially. **Diction** marks all pattern matches in a sentence with brackets ([ ]). Although many of the phrases in the default data base may be correct in some contexts, they generally indicate an awkward or verbose construction. Some examples of the phrases and suggested alternatives are:

Phrase:	Alternative:
a large number of	many
arrive at a decision	decide
collect together	collect
for this reason	so
pertaining to	about
through the use of	by or with
utilize	use
with the exception of	except

All of the following examples contain the repetitious and awkward phrase "the fact":

Phrase:	Alternative:
accounted for by the fact that	caused by
an example of this is the fact that	thus
based on the fact that	because
despite the fact that	although
due to the fact that	because
in light of the fact that	because
in view of the fact that	since
not withstanding the fact that	although

If you have some phrases that you particularly dislike, or feel you use too often, you may create your own file of patterns. Then, you can invoke the **diction** program with the **-f** option:

    diction -f *patternfile*

The default pattern file for the **diction** program will be loaded first, followed by your pattern file. In this way, you can either suppress patterns contained in the default file or include your own favorites in addition to those in the default file. You can also use the **-n** option to exclude the default file altogether:

    diction -n patternfile

In constructing a pattern file, spaces should be used before and after each phrase to avoid matching substrings in words. For example, to find all occurrences of the word "the", use leading and trailing spaces, so that only the word "the" is matched and not the string "the" in words like there, other, and therefore. Note however, that one side effect of surrounding the words with spaces is that if two instances occur without intervening words, e.g., "the the", only the first will be matched because the intervening space will be counted as part of the first pattern.

Getting Started with MM

This chapter provides a simple introduction to **MM**, the "Memorandum Macros", a macro package which you can use on your XENIX System with either of the two XENIX formatting programs, **nroff** or **troff**, to produce formatted text for the lineprinter or typesetter, respectively. The features of **MM** are described comprehensively in the next chapter, "MM Reference". You can learn to use the **MM** macros quickly and format text immediately, without learning the more complicated **nroff** or **troff** formatting commands.

The **MM** program reads the commands you have inserted in your text and "translates" them into **nroff** or **troff** commands at the time your text file is processed. With **MM** you can specify the style of paragraphs, section headers, lists, page numbering, titles, and footnotes. You can also produce cover pages, abstracts, and tables of contents, as well as control font changes and multicolumn output. If you are using **MM** along with **troff** to output your text to a phototypesetter, you can specify variable spacing and the size of your type.

Although using **nroff** or **troff** directly offers you a much wider range of commands and options, we recommend that you use **MM** for most of your formatting needs. Use the **nroff** and **troff** requests discussed in Chapter 5, "Using Nroff/Troff" and Chapter 6, "Nroff/Troff Reference" only when necessary.

Inserting MM Macros

To use the **MM** macros to format a document, type in your text normally, interspersed with formatting commands. These commands are uppercase letters preceded by a dot (.) and appear at the beginning of a line. Instead of indenting for paragraphs, for example, you use the .P macro before each paragraph, to produce extra line space:

    .P
    To meet the objectives proposed at the meeting...

The .P macro can also be used to indent paragraphs. For more information, see the section on "Paragraphs" in Chapter 4.

A single **MM** macro can often perform a number of formatting functions at once. In a long document, you might have several sections, each beginning with a numbered heading, like this

1.0 Saltwater Fishing in the Pacific Northwest

To create this header, you would enter:

    .H 1 "Saltwater Fishing in the Pacific Northwest"

Not only will **MM** create a bold heading and leave a space between the heading and the text which follows, it will also automatically number all the headings in the document sequentially. Furthermore, if you use the table of contents macro (.TC) at the end of the document, **MM** will create a table of contents, listing all the numbered headings and the pages where they occur.

The **MM** macros also provide a convenient facility for creating a consistent format for such document elements as lists and displays. For example, if you wanted a "bullet list" to look like this:

- Convenience

- Ease of use

- Portability

you would enter the following text and macros:

```
.BL
.LI
Convenience
.LI
Ease of use
.LI
Portability
.LE
```

MM will provide the current indents, spacing, and bullets.

Note that you must always begin a document to be formatted with **MM** with a macro, rather than an ordinary line of text. You might just start with a .P command, for example, to begin your document with an ordinary paragraph.

Invoking MM

After you have created a file containing text and **MM** macros, you can format it with the following command:

```
nroff -mm filename>filename.mm&
```

This command line tells the XENIX system that you want to format the document, using the **nroff** formatting program, and the **MM** macro package to prepare it for a letter-quality printer or lineprinter. Once the document is formatted, it will be stored in *filename.mm* or whatever file you specify. You can then send it to the lineprinter with the command:

```
lpr filename.mm
```

If you are formatting documents for printing on a typesetter, you would use the **MM** macros with the **troff** program instead:

```
troff -mm filename>filename.t
```

If you have more complex formatting, such as two-column text, formatted with the two-column (.2C) macro, or if you have used the table start (.TS) and table end (.TE) macros to produce multicolumn tabular material, you must remember to pipe the output through the preprocessor **col,** in order to prepare the columns of text. Your command line might look like this:

    nroff -mm filename I col>filename.mm

If you are using the **tbl** or **eqn** programs to produce tables and mathematical equations, you must also process the files through these programs first, using **tbl** before **eqn,** as in the following:

    tbl filenameIneqnInroff -mm >filename.mm

Basic Formatting Macros

The following sections describe the most commonly used **MM** macros, including macros to define paragraphs, headings, lists, font changes, displays, and tables. For more detailed information about each of these macros, read Chapter 4, "MM Reference".

Paragraphs and Headings

With **MM,** it is easy to specify paragraph and heading style. For example, look at the following passage:

1.0 Paragraphs and Headings

This section describes the types of paragraphs and the kinds of headings that are available.

1.1 Paragraphs

Paragraphs are specified with the .P macro. Usually, they are flush left.

1.2 Headings

Numbered Headings
There are seven levels of numbered headings. Level 1 is the highest; level 7 is the lowest.

Headings are specified with the .H macro, whose first argument is the level of heading (1 through 7).

Unnumbered Headings
The macro .HU is a special case of .H which creates a heading with no heading number.

To create this heading format, you would insert the following in a text file:

```
.H 2 "Paragraphs and Headings"
This section describes the types of paragraphs and the
kinds of headings that are available.
.H 3 "Paragraphs"
Paragraphs are specified with the .P macro. Usually, they are
flush left.
.H 3 "Headings"
.HU "Numbered Headings"
There are seven levels of numbered headings. Level 1 is the
highest; level 7 is the lowest.
.P
Headings are specified with the .H macro, whose first argument is
the level of heading (1 through 7).
.HU "Unnumbered Headings"
The macro .HU is a special case of .H which creates a heading
with no heading number.
```

MM produces these headings in default styles which can be redefined, if necessary. This is described in detail in Chapter 4, "MM Reference". The headings are automatically numbered and are used to print a table of contents, if the table of contents (.TC) macro is used. The numbers may be altered or reset with the number register (.nr) request. To restart the numbering of a second level heading at 1, you would insert the following command:

    .nr H2 1

Lists

All list formats in **MM** have a list-begin macro, one or more list items, each consisting of a .LI macro followed by the list item text, and the list-end macro (.LE). In addition to the bullet list demonstrated at the beginning of this chapter, there is also the dash list, using the list begin macro (.DL) to create a list format like the bullet list except marked with dashes rather than bullets. A mark list (.ML) is also available, to mark list items with the character of your choice.

The automatic list (.AL) macro automatically numbers list items in one of several ways. When specified alone, or followed by "1", the .AL macro numbers the list items with Arabic numbers. The macro .AL A specifies a list ordered A, B, C, etc. The macro .AL followed by a lowercase a (.AL a), specifies a, b, c, etc. The macro .AL I numbers list items with Roman numerals. .AL i numbers a list with lowercase Roman numerals (i, ii, iii, etc.)

Numbered lists may be nested to produce outlines and other formats. For example:

    I.   Incan Archaeological Sites

        A.   Peru

            1.   Macchu Picchu

            2.   Pisac

        B.   Ecuador

This is produced with:

```
.AL I
 Incan Archaeological Sites
.AL A
.LI
Peru
.AL 1
.LI
Macchu Picchu
.LI
Pisac
.LE
.LI
Ecuador
.LE
.LE
```

In addition to the numbered and marked lists, **MM** offers a variable list (.VL) macro, which is useful for producing two-column lists with indents. The .VL macro is described in detail in Chapter 4, "MM Reference".

Font Changes and Underlining

To produce italics on the typesetter, precede the text to be italicized with the sequence \fI and follow it with \fR. For example:

```
\fIas much text as you want
can be typed here\fR
```

Italics are represented on lineprinters and letter-quality printers by underlining. The .R command restores the normal (usually Roman) font.

If only one word is to be italicized, it may be typed alone on a line after a .I command:

```
.I word
```

In this case no .R is needed to restore the previous font. The default font is automatically restored on the next line.

Similarly boldface can be produced by typing:

```
.B
Text to be set in boldface
goes here
.R
```

As with the .I macro, a single word can be placed in boldface by placing it alone on the same line with a .B command.

Footnotes

Material placed between lines with the footnote start (.FS) and footnote end (.FE) macros will be collected, and placed at the bottom of the current page. The footnotes are automatically numbered, or an optional footnote mark may be used. This mark follows the .FS macro. For example:

```
Without further research
.FS
As demonstrated by Tiger and Leopard (1975).
.FE
the claim could not be substantiated.
However, other studies
.FS *
For example, Panther and Lion (1981).
.FE
indicated that the correlation was significant.
```

produces one numbered footnote:

        1. As demonstrated by Tiger and Leopard (1975).

and one marked footnote:

        * For example, Panther and Lion (1981).

Displays and Tables

To prepare displays of lines, such as tables, which are to be set off from the running text, enclose them in the commands .DS and .DE. For example:

```
.DS
text goes here
.DE
```

By default, lines between .DS and .DE are left-adjusted. You may also specify a left-adjusted display by using .DS L. To get an indented display, use .DS I. You can also create centered tables with .DS C. For example:

                This is a centered display preceded by
                a .DS C and followed by a .DE command.

or:

This is a left-adjusted display
preceded by a .DS L command
and followed by a .DE command.

Note that the .DS C macro centers each subsequent line. You can also use .DS B to make the display into a left-adjusted block of text and then center that entire block. Normally a display is kept together on one page. Text within a display is produced in "nofill" mode, i.e., lines of text are not rearranged.

Memos

If you need to produce many memos in a standardized format, you may find the memorandum type (.MT) type macros useful for creating titling information. Be warned, however, that because these memorandum types were originally developed inside Bell Laboratories, some of the possible parameters to this macro automatically print the string "Bell Laboratories" on the memo. To suppress this, be sure to use the "affiliation" (.AF) macro after an .MT macro, as follows:

      .AF " "

or, if you wish to have your own company or organization name appear automatically, use:

      .AF "Widgets, Ltd."

There are a number of parameters which substantially change the format and content of memoranda output and it is critical that you insert the macros in the correct order. Therefore, it is important that you read the section on "Memorandum Type" in Chapter 4, "MM Reference" before inserting these macros. Once you have chosen the appropriate type, you should be able to reuse these macros for all your memos to produce a standard style.

Multicolumn Formats

If you place the command .2C in your document, the document will be printed in double column format beginning at that point. This feature is generally not accommodated by an ordinary lineprinter, but is often desirable on the typesetter. The command .1C stops two-column output and returns to one-column output.

Using Nroff/Troff Commands

If you want to format text using **MM** without learning the other formatting programs, you should become familiar with at least a few simple **nroff/troff** commands, which you will probably need to supplement the **MM** macros. These work with both typesetter and lineprinter or terminal output:

.bp   Begin new page.

.br   "Break", that is, stop running text from line to line.

.sp n   Insert n blank lines.

Checking MM Input with mmcheck

The program **mmcheck** can be used to check the accuracy of your input to **MM,** without actually formatting a document. If you use **mmcheck** regularly, you will save a great deal of processing time, because you will be able to "debug" your input file quickly, without running the **nroff** and **troff** programs. To invoke **mmcheck,** use the command line:

mmcheck *filename*

The output of by default. **Mmcheck** checks for correct pairing of macros, including .DS/.DE, .TS/.TE, and .EQ/.EN. It also looks for list specification format, making sure that every list has a list begin macro (.AL, .DL, .BL, .ML, .VL, etc.) and a list end macro (.LE). Normally, **mmcheck** prints a list of errors and the lines where they occurred. For example:

chap1.s:
        Extra .DE at line 74
        539 lines done.

Note, however, that the location of an error may occasionally be obscured. In the example above, the "extra" .DE could actually be caused by a missing .DS.

CHAPTER 4
MM REFERENCE

Introduction

This chapter is the reference guide for the **MM** Memorandum Macros. **MM** provides a unified, consistent, and flexible tool for producing many common types of documents, often eliminating the need for working directly with **nroff** or **troff** commands. **MM** is the standard, general-purpose macro package for most documents.

Using the **MM** macros, you can produce letters, reports, technical memoranda, papers, manuals, and books. Documents may range in length from single-page letters to documents that are hundreds of pages long.

Why Use MM?

There are several reasons why we recommend using **MM**, instead of working with the formatting programs **nroff** and **troff** directly. These include:

- You need not be an expert to use **MM** successfully. If your input is incorrect, the macros attempt to interpret it, or a message describing the error is output.

- Reasonable default values are provided so that simple documents can be prepared without complex sequences of commands.

- Parameters are provided to allow for individual preferences and requirements in document styling.

- The capability exists for expert users to extend the **MM** macros by adding new macros or redefining existing ones.

- The output of **MM** is device independent, allowing the use of terminals, lineprinters, and phototypesetters with no change to the macros.

- The need for repetitious input is minimized by allowing the user to specify parameters once at the beginning of a document.

- Output style can be modified without making changes to the document input.

Organization and Conventions

Each section of this chapter explains a feature of **MM**, with the more commonly used features explained first. You may find you have no need for the information in the later sections, or for some of the options and parameters which accompany even common features. This reference guide is organized so that you can skim a section to obtain formatting information you need, and skip features for which you have no use.

Structure of a Document

Input for a document to be formatted with **MM** contains four major parts, any of which are optional. If present, they must occur in the following order:

1. Parameter-setting. This segment determines the general style and appearance of a document, including page width, margin justification, numbering styles for headings and lists, page headers and footers, and other properties. In this segment, macros can be added or redefined. If omitted, **MM** will produce output in a default format; this segment produces no actual output, but performs the setup for the rest of the document.

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

3. Body. This segment contains the actual text of the document. It may be as small as a single paragraph, or as large as hundreds of pages. It may include hierarchically-ordered headings of up to seven levels, which may be automatically numbered and saved to generate the table of contents. Also available are list formats with up to five levels of subordination, which may have automatic numbering, alphabetic sequencing, and marking. The body may contain various types of displays, tables, figures, references, and footnotes.

4. Ending. This segment contains those items that occur only once at the end of a document. Included here are signature(s) and lists of notations (e.g., "copy to" lists). In this segment, macros may be invoked 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.

The size or existence of any of these segments depends on the type and length of the document. Although a specific item (such as date, title, author's name) may be printed in several different ways depending on the document type, it will always be entered in the same form.

Definitions

The following terms are used throughout this chapter:

Formatter   Refers to either of the text-formatting programs **nroff** or **troff.**

Requests   Built-in commands recognized by the formatters. Although it may not be necessary to use these requests directly, they are referred to in this chapter.

Macros   Named collections of requests. Each macro is an abbreviation for a
                    collection of requests that would otherwise require repetition. **MM**
                    supplies many predefined macros, and you may define additional
                    macros as necessary. Macros and requests share the same set of names
                    and are used in the same way.

Strings   Provide character variables, each of which names a string of
                    characters. Strings are often used in page headers, page footers, and
                    lists. They use the same names as requests and macros. A string can
                    be defined with the define string (.ds) request, and then referred to by
                    its name, preceded by * for a one-character name or *(for a two-
                    character name.

Number registers
                    Integer variables used for flags, arithmetic, and automatic numbering.
                    A register can be given a value using a number register (.nr) request,
                    and can be referenced by preceding its name by \n for one-character
                    names or \n(for two-character names.

Invoking the Macros

This section describes the command lines necessary to **MM** with different options on
various output devices.

The MM Command

The **MM** command is used to print documents using **nroff** and **MM**. This command is
equivalent to invoking **nroff** with the −**mm** flag. Options are available to specify
preprocessing by **tbl** and/or by **eqn/neqn,** and for postprocessing by various output
filters, such as **col.** Any arguments or flags not recognized by **MM** are passed to **nroff.**
The following options can occur in any order before the filenames:

−e   Invokes **neqn.**

−t   Invokes **tbl.**

−c   Invokes **col.**

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

−y   Invokes −mm (uncompacted macros) instead of −cm (see next section).

−12   Invokes 12-pitch mode (the pitch switch on the terminal must be set to 12).

The -cm or -mm Flags

The **MM** package can also be invoked by including the –cm or –mm flag as an argument to the formatter, as in:

    nroff -mm file

Typical Command Lines

The prototype command lines are as follows:

Text without tables or equations:

    mm [options] *filename*
    nroff [options] *filename*
    troff [options] *filename*

Text with tables:

    mm -t [options] *filename*
    tbl *filename*|nroff [options] -mm
    tbl *filename*|troff [options] -mm

Text with equations:

    mm -e [options] *filename*
    neqn *filename*|nroff [options] -mm
    eqn *filename*|troff [options] -mm

Text with both tables and equations:

    mm -t -e [options] *filename*
    tbl *filename*|neqn|nroff [options] -mm
    tbl *filename*|eqn|troff [options] -mm

If two-column processing is used with **nroff,** either the –c option must be specified to **MM** or the **nroff** output must be postprocessed by **col.**

Command Line Parameters

Number registers hold parameter values that control various aspects of output style. Many of these can be changed within the text files with number register (.nr) requests. In addition, some of these registers can be set from the command line itself, a useful

feature for those parameters that should not be permanently embedded within the input text itself. If used, these registers must be set on the command line or before the **MM** macro definitions are processed. These are:

-rA*n*	For *n* = 1, this has the effect of invoking the .AF macro without an argument.
-rC*n*	*N* sets the type of copy (e.g., DRAFT) to be printed at the bottom of each page:

        n = 1   For OFFICIAL FILE COPY

        n = 2   For DATE FILE COPY

        n = 3   For DRAFT with single-spacing and default paragraph style

        n = 4   For DRAFT with double-spacing and 10-space paragraph indent

-rD1	Sets "debug mode". This flag requests the formatter to continue processing even if **MM** detects errors that would otherwise cause termination. It also includes some debugging information in the default page header.
-rE*n*	Controls the font of the Subject/Date/From fields. If n = 0 these fields are bold (default for **troff**) and if n = 1 they are regular text (default for **nroff**).
-rL*k*	Sets the length of the physical page to *k* lines. For **nroff**, *k* is an unscaled number representing lines or character positions; for **troff**, *k* must be scaled. The default value is 66 lines per page.
-rN*n*	Specifies the page numbering style. When *n* = 0 (default), all pages get the (prevailing) header. When *n* = 1, the page header replaces the footer on page 1 only. When *n* = 2, the page header is omitted from page 1. When *n* = 3, section-page numbering occurs. When *n* = 4, the default page header is suppressed, but user-specified headers are not affected. When *n* = 5, section-page and section-figure numbering occurs.

        The contents of the prevailing header and footer do not depend on the value of the number register N; N only controls whether and where the header (and, for N = 3 or 5, the footer) is printed, as well as the page numbering style. In particular, if the header and footer values are null, the value of N is irrelevant.

-rO*k*	Offsets output *k* spaces to the right. For **nroff**, these values are unscaled numbers representing lines or character positions. For **troff**, these values must be scaled. This register is helpful for adjusting output positioning on some terminals. If this register is not set on the command line the default offset is .75 inches. NOTE: The register name is the capital letter (O), not the digit zero (0).

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

-rS*n*   Sets the point size and vertical spacing. The default *n* is 10, i.e., 10-point type on 12-point leading (vertical spacing), giving 6 lines per inch. This parameter applies to **troff** only.

-rT*n*   Provides register settings for certain devices. If *n* =1, then the line length and page offset are set to 80 and 3, respectively. Setting *n* to 2 changes the page length to 84 lines per page and inhibits underlining. The default value for *n* is 0. This parameter applies to **nroff** only.

-rU1   Controls underlining of section headings. This flag causes only letters and digits to be underlined. Otherwise, all characters (including spaces) are underlined. This parameter applies to **nroff** only.

-rW*k*   Sets page width (i.e., line length and title length) to *k*. For **nroff**, *k* is an unscaled number representing lines or character positions; for **troff**, *k* must be scaled. This register can be used to change the page width from the default value of 6.0 inches (60 characters in 10 pitch or 72 characters in 12 pitch).

Omission of -cm or -mm

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

        .ss 18
        .so /usr/lib/tmac/tmac.m
        .ss 12
        remainder of text

In this case, do not use the –**cm** or –**mm** flags (or the **MM** or **mmt** commands); the .so request has the equivalent effect. The registers must be initialized before the .so request, because their values are meaningful only if set before the macro definitions are processed. When using this method, it is best to put into the input file only those parameters that are seldom changed. For example:

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

specifies, for **nroff,** a line length of 80, a page offset of 10, and section-page numbering.

Formatting Concepts

The normal action of the formatters is to fill output lines from one or more input lines. The output lines may be justified so that both the left and right margins are aligned. As the lines are being filled, words may also be hyphenated as necessary. It is possible to turn any of these modes on and off. Turning off fill mode also turns off justification and hyphenation.

Certain formatting commands (both requests and macros) cause the filling of the current output line to cease. Printing of a partially filled output line is known as a "break". A few formatter requests and most of the **MM** macros cause a break.

While formatter requests can be used with **MM,** they occasionally have unpredicted consequences. There should be little need to use formatter requests. The macros described in this section should be used in most cases because you will be able to control and change the overall style of the document easily and specify complex features, such as footnotes or tables of contents, without using intricate formatting requests. A good rule is to use **nroff** and **troff** requests directly only when absolutely necessary.

To make future revision easier, input lines should be kept short and should be broken at the end of clauses; each new full sentence should begin on a new line.

Arguments and Quoting

For any macro, a "null argument" is an argument whose width is zero. Such an argument often has a special meaning; the preferred form for a null argument is double quotation marks ("). Omitting an argument is not the same as supplying a null argument. Furthermore, omitted arguments can occur only at the end of an argument list, while null arguments can occur anywhere.

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

Double quotation marks (") are not permitted as part of the value of a macro argument or of a string that is to be used as a macro argument. If you must, use two grave accents (` `) and/or two acute accents (' ') instead. This restriction is necessary because many macro arguments are processed (interpreted) several times. For example, headings are first printed in the text and may be reprinted in the table of contents.

Unpaddable Spaces

When output lines are justified to give an even right margin, existing spaces in a line may have additional spaces appended to them. This may affect the desired alignment of text. To avoid this problem, it is necessary to be able to specify a space that cannot be expanded during justification, i.e., an "unpaddable space". There are several ways to do this. First, you may type a backslash (\) followed by a space. This pair of characters generates an unpaddable space. Second, you may sacrifice some seldom-used character to be translated into a space upon output. Because this translation occurs after justification, the chosen character may be used anywhere an unpaddable space is desired. The tilde () is often used for this purpose. To use it in this way, insert the following line at the beginning of the document:

        .tr

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

        .t r

before the place where it is needed. Its previous usage is restored by repeating the .tr , but only after a break or after the line containing the tilde has been forced out. Use of the tilde in this way is not recommended for documents in which the tilde is used within equations.

Hyphenation

The formatters do not perform hyphenation unless the user requests it. Hyphenation can be turned on in the body of the text by specifying

        .nr Hy 1

at the beginning of the document. The "1" is a special code that means to turn hyphenation on. If hyphenation is requested, the formatters will automatically hyphenate words as needed. However, you may specify the hyphenation points for a specific occurrence of any word by using a special character known as a "hyphenation indicator" (initially, the two-character sequence \%), or you may specify hyphenation points for a small list of words (about 128 characters).

If the hyphenation indicator (initially, the two-character sequence \%) appears at the beginning of a word, the word is not hyphenated. It can also be used to indicate legal hyphenation point(s) inside a word. In any case, all occurrences of the hyphenation indicator disappear on output.

The user may specify a different hyphenation indicator with the command:

    .HC [hyphenation-indicator]

The caret (^) is often used for this purpose; this is done by inserting the following at the beginning of a document:

    .HC ^

Note that any word containing hyphens or dashes--also known as em dashes--will be broken immediately after a hyphen or dash if it is necessary to hyphenate the word, even if the formatter hyphenation function is turned off.

Using the .hw request, you may supply a small list of words with the proper hyphenation points indicated. For example, to indicate the proper hyphenation of the word "printout", you may specify:

    .hw print-out

For more information about hyphenation, see "Hyphenation" in Chapter 6.

Tabs

The macros .MT, .TC, and .CS use the .ta request to set tab stops, and then restore the default values of tab settings. Setting tabs to other than the default values is the user's responsibility.

Note that 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. The **tbl** program changes tab stops but does not restore the default tab settings.

Bullets

A bullet (•) is often obtained on a typewriter terminal by using the letter o overstruck by a +. For compatibility with **troff**, a bullet string is provided by **MM**. Rather than overstriking, use the sequence:

    *(BU

wherever a bullet is desired. Note that the bullet list (.BL) macro uses this string to automatically generate bullets for the list items.

Dashes, Minus Signs, and Hyphens

Troff has distinct graphics for a dash, a minus sign, and a hyphen, while **nroff** does not. If you intend to use **nroff** only, you can use the minus sign (-) for all three.

If you plan to use both formatters, you must be careful in preparing text. Unfortunately, these characters cannot be represented in a way that is both compatible and convenient. Try the following:

Dash   Use *(EM for each text dash for both **nroff** and **troff**. This string generates an em dash (--) in **troff** and two dashes (--) in **nroff**. Note that the dash list (.DL) macro automatically generates the em dashes for the list items.

Hyphen   Use the hyphen character (-) for both formatters. **Nroff** will print it as is, and **troff** will print a true hyphen.

Minus   Use \- for a true minus sign, regardless of formatter. **Nroff** will ignore the \, while **troff** will print a true minus sign.

Trademark String

The trademark string *(Tm places the letters TM one half-line above the text that it follows. For example, the input:

    The XENIX*(Tm System Reference Manual.

yields:

    The XENIX™ System Reference Manual.

Paragraphs and Headings

This section describes simple paragraphs and section headings.

Paragraphs

The paragraph macro is used to begin two kinds of paragraphs:

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

The type is p, i, or s, as explained in the following paragraphs.

In a "left-justified" paragraph, the first line begins at the left margin, while in an "indented" paragraph, it is indented five spaces.

A document has a default paragraph style obtained by specifying .P before each paragraph that does not follow a heading. The default style is controlled by the number register Pt. The initial value of Pt is 0, which always provides left-justified paragraphs. All paragraphs can be forced to be indented by inserting the following at the beginning of the document:

    .nr Pt 1

All paragraphs will be indented except after headings, lists, and displays if the following:

    .nr Pt 2

is inserted at the beginning of the document.

The amount a paragraph is indented is contained in the register Pi, whose default value is 5. To indent paragraphs by 10 spaces, for example, insert:

    .nr Pi 10

at the beginning of the document. Both the Pi and Pt register values must be greater than zero for any paragraphs to be indented.

The number register Ps controls the amount of spacing between paragraphs. By default, the Ps register is set to 1, yielding one blank space (1/2 vertical space). Values that specify indentation must be unscaled and are treated as "character" positions, i.e., as a number of ens. In **troff**, an en is the number of points (1 point = 1/72-inch) equal to half the current point size. In **nroff**, an en is equal to the width of a character.

Regardless of the value of Pt, an individual paragraph can be forced to be left-justified or indented. .P always forces left justification; .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.

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

A different style of numbered paragraphs is obtained by using the

    .nP

macro rather than the .P macro for paragraphs. This produces paragraphs that are numbered within second level headings and contain a double-line indent in which the text of the second line is indented to be aligned with the text of the first line so that the number stands out. For example:

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

Numbered Headings

The heading macro has the form:

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

The .H macro provides seven levels of numbered headings. Level 1 is the highest; level 7 the lowest. The *heading-suffix* is appended to the *heading-text* and may be used for footnote marks which should not appear with the heading text in the table of contents. You will not need to insert a .P macro after a .H or .HU macro, because the .H macro also performs the function of the .P macro. If a .P follows a .H, the .P is ignored.

The effect of .H varies according to the level argument. First-level headings are preceded by two blank lines (one vertical space); all others are preceded by one blank line.

.H 1 *heading-text*
> Gives a bold heading followed by a single blank line. The following text begins on a new line and is indented according to the current paragraph type. Full capital letters should normally be used to make the heading stand out.

.H 2 *heading-text*
> Yields a bold heading followed by a single blank line. The following text begins on a new line and is indented according to the current paragraph type. Normally, initial capitals are used.

.H n *heading-text*
> Where n is a number greater than 3 and less than 7, produces an underlined (italic) heading followed by two spaces. The following text appears on the same line.

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

Appearance of Headings

You can modify the appearance of headings quite easily by setting certain registers and strings at the beginning of the document. In this way you can quickly alter a document's style because the style control information is concentrated in a few lines, rather than distributed throughout the document.

A first-level heading normally has two blank lines (one vertical space) preceding it, and all others have one blank line. If a multiline heading splits across pages, it is automatically moved to the top of the next page. Every first-level heading may be forced to the top of a new page by inserting:

    .nr Ej 1

at the beginning of the document. Long documents may be made more manageable if each section starts on a new page. Setting Ej to a higher value has the same effect for headings up to that level, i.e., a page eject occurs if the heading level is less than or equal to Ej.

Three registers control the appearance of text immediately following an .H macro. They are heading break level (Hb), heading space level (Hs), and post-heading indent (Hi).

If the heading level is less than or equal to Hb, a break occurs after the heading. If the heading level is less than or equal to Hs, a blank line is inserted after the heading. Defaults for Hb and Hs are 2. If a heading level is greater than Hb and also greater than Hs, then the heading (if any) is run into the following text. With these registers, you can separate headings from text consistently throughout the document, and allow for easy alteration of whitespace and header emphasis.

For any stand-alone heading, i.e., a heading not run into the following text, the alignment of the next line of output is controlled by the register Hi. 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 register Pt. Finally, if Hi is 2, text is indented to line up with the first word of the heading itself, so that the heading number stands out more clearly.

For example, to cause a blank line to appear after the first three heading levels, to have no run-in headings, and to force the text following all headings to be left-justified (regardless of the value of Pt), the following lines should appear at the top of the document:

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

The register Hc can be used to obtain centered headings. A heading is centered if its level is less than or equal to Hc, and if it is stand-alone. Hc is 0 by default (no centered headings).

Bold, Italic, and Underlined Headings

Any heading that is underlined by **nroff** is made italic by **troff.** The string HF (heading font) contains seven codes that specify the fonts for heading levels 1-7.

Levels 1 and 2 are bold; levels 3 through 7 are underlined in **nroff** and italic in **troff.** The user may reset HF as desired. Any value omitted from the right end of the list is taken to be 1. For example, the following would result in five bold levels and two nonunderlined (Roman) levels:

    .ds HF 3 3 3 3 3

Nroff can underline in two ways. The underline (.ul) request underlines only letters and digits. The continuous style (.cu) request underlines all characters, including spaces. By default, **MM** attempts to use the continuous style on any heading that is to be underlined and is short enough to fit on a single line. If a heading is too long, only letters and digits are underlined.

Using the -rU1 flag when invoking **nroff** forces the underlining of only letters and digits in all headings.

Heading Point Sizes

If you are using **troff,** you may specify the desired point size for each heading level with the HP string, as follows:

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

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

    .ds HP 12 12 11 10 10 10 10

prints the first two heading levels in 12-point type, the third heading level in 11-point type, and the remainder in 10-point type. The specified values may also be relative point-size changes, e.g.:

    .ds HP +2 +2 -1 -1

If absolute point sizes are specified, those sizes will be used regardless of the point size of the body of the document. If relative point sizes are specified, then the point sizes for the headings will be relative to the point size of the body, even if the point size of the body is changed. Omitted or zero values imply that the default point size will be used for the corresponding heading level.

Note

> When you change the point size of headings, vertical spacing remains unchanged. Therefore, if you specify a large point size for a heading, you must also increase vertical spacing (with .HX and/or .HZ) to prevent overprinting.

Marking Styles

The heading mark macro has the form:

    .HM [arg1] ...[arg7]

to change the heading mark style of a heading. The registers named H1 through H7 are used as counters for the seven levels of headings. Their values are normally printed using Arabic numerals. The heading mark style (.HM) macro allows this choice to be overridden. This macro can have up to seven arguments; each argument is a string indicating the type of marking to be used. Omitted values are interpreted as 1; illegal values have no effect. The values available are:

Value	Interpretation
1	Arabic (default for all levels)
0001	Arabic with enough leading zeroes to get specified digits
A	Uppercase alphabetic
a	Lowercase alphabetic
I	Uppercase Roman
i	Lowercase Roman

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

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

    .HM I A 1 a i
    .nr Ht 1

Unnumbered Headings

The unnumbered heading macro has the form:

        .HU heading-text

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

        .HU heading-text

and

        .H 2 heading-text

is the printing of the heading mark for the latter. Both have the effect of incrementing the numbering counter for level 2, and resetting to zero the counters for levels 3 through 7. Typically, the value of Hu should be set to make unnumbered headings (if any) be the lowest-level headings in a document. .HU can be especially helpful in setting up appendices and other sections that may not fit well into the numbering scheme of the main body of a document.

Headings and the Table of Contents

The text of headings and their corresponding page numbers can be automatically collected for a table of contents. This is accomplished by specifying in the register Cl what level headings are to be saved, then invoking the .TC macro at the end of the document.

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

Because of the way the headings are saved, it is possible to exceed the formatter's storage capacity, particularly when saving many levels of many headings while also processing displays and footnotes. If this happens, an "Out of temp file space" message will occur; the only remedy is to save fewer levels or to have fewer words in the heading text.

First-Level Headings and the Page Numbering Style

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

User Exit Macros

This section is intended only for users who are accustomed to writing formatter macros. With .HX, .HY and .HZ you can obtain control over the previously described heading macros. You must define these macros yourself and use them in the form:

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

The .H macro invokes .HX shortly before the actual heading text is printed; it calls .HZ as its last action. After .HX is invoked, the size of the heading is calculated. This processing causes certain features that may have been included in .HX, such as .ti for temporary indent, to be lost. After the size calculation, .HY is invoked so that you may specify these features again. All the default actions occur if these macros are not defined. If you define .HX, .HY, or .HZ, your definition is interpreted at the appropriate point. These macros can therefore influence the handling of all headings, because the .HU macro is actually a special case of the .H macro.

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

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

string }0
> If *rlevel* is nonzero, this string contains the heading mark. If *rlevel* is 0, this string is null.

register ;0
> This register indicates the type of spacing that is to follow the heading. A value of 0 means that the heading is run-in. A value of 1 means a break (but no blank line) is to follow the heading. A value of 2 means that a blank line is to follow the heading.

string }2
> If register ;0 is 0, this string contains two unpaddable spaces that will be used to separate the heading from the following text. If register ;0 is nonzero, this string is null.

register ;3
> This register contains an adjustment factor for an .ne request issued before the heading is actually printed. On entry to .HX, it has the value 3 if *dlevel* equals 1, and 1 otherwise. The .ne request is for the following number of lines: the contents of the register ;0 taken as blank lines (halves of vertical space), plus the contents of register ;3 as blank lines (halves of vertical space) plus the number of lines of the heading.

The user may alter the values of }0, }2, and ;3 within .HX as desired. If you use temporary string or macro names within .HX, choose them carefully.

.HY is called after the .ne is issued. Certain features requested in .HX must be repeated. For example:

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

.HZ is called at the end of .H to permit user-controlled actions after the heading is produced. For example, in a large document, sections may correspond to chapters of a book, and you may want to change a page header or footer. For example:

```
.de HZ
.if \\$1=1 .PF " `  `Section \\$2'  ' "
.P
..
```

Lists

This section describes the kinds of lists which can be obtained with the **MM** macros, including automatically numbered and alphabetized lists, bullet lists, dash lists, lists with arbitrary marks, and lists starting with arbitrary strings (e.g., with terms or phrases to be defined).

In order to avoid repetitive typing of arguments to describe the appearance of items in a list, **MM** provides a convenient way to specify lists. All lists are composed of the following parts:

● A "list-initialization" macro that controls the appearance of the list (e.g. line spacing, indentation, marking with special symbols, and numbering or alphabetizing).

- One or more "list item" macros, each followed by the actual text of the corresponding list item.

- The "list end" macro that terminates the list and restores the previous indentation.

Lists may be nested up to five levels. The list-initialization (.LI) macro saves the previous list status (e.g., indentation, marking style, etc.); the list-end (.LE) macro restores it. The format of a list is specified only once at the beginning of list. You may also create your own customized sets of list macros with relatively little effort.

Sample Nested List

The input for several lists and the corresponding output are shown below. The .AL and .DL macros are examples of the "list-initialization" macros. Here is some sample input text:

```
.AL A
.LI
This is an alphabetized item.
This text shows the alignment of the second line of the item.
The quick brown fox jumped over the lazy dog's back.
.AL
.LI
This is a numbered item.
This text shows the alignment of the second line of the item.
The quick brown fox jumped over the lazy dog's back.
.DL
.LI
This is a dash item.
This text shows the alignment of the second line of the item.
The quick brown fox jumped over the lazy dog's back.
.LI + 1
This is a dash item with a plus as prefix.
This text shows the alignment of the second line of the item.
The quick brown fox jumped over the lazy dog's back.
.LE
.LI
This is numbered item 2.
.LE
.LI
This is another alphabetized item, B.
This text shows the alignment of the second line of the item.
The quick brown fox jumped over the lazy dog's back.
.LE
.P
This paragraph appears at the left margin.
```

The output looks like this:

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

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

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

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

        2.   This is numbered item 2.

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

This paragraph appears at the left margin.

List Item

The list item macro has the form:

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

The .LI macro is used with all lists. It normally causes the output of a single blank line before its item, although this may be suppressed. If no arguments are given, it labels its item with the "current" mark which is specified by the most recent list-initialization macro. If a single argument is given to .LI, that argument is output instead of the current mark. If two arguments are given, the first argument becomes a prefix to the current mark, thus allowing you to emphasize one or more items in a list. One unpaddable space is inserted between the prefix and the mark. For example:

```
.BL
.LI
This is a simple bullet item.
.LI +
This replaces the bullet with a plus.
.LI + 1
But this uses plus as prefix to the bullet.
.LE
```

This yields:

- This a simple bullet item.

+ This replaces the bullet with a plus.

+• But this uses plus as prefix to the bullet.

Note that the mark must not contain ordinary (paddable) spaces, because alignment of items will be lost if the right margin is justified. If the "current mark" in the current list is a null string, and the first argument of .LI is omitted or null, the resulting effect is that of a "hanging indent", i.e., the first line of the following text is outdented, starting at the same place where the mark would have started.

List End

The list end macro has the form:

      .LE [1]

The list end macro restores the state of the list to that existing just before the most recent list-initialization macro call. If the optional argument is given, the .LE outputs a blank line. You should use this option only when the .LE is followed by running text, but not when followed by a macro that produces blank lines of its own, such as .P, .H, or .LI.

.H and .HU automatically clear all list information, so you may omit the .LE(s) that would normally occur just before either of these macros. This is not recommended, however, because errors will occur if the list text is separated from the heading at some later time (e.g., by insertion of text).

Initializing Automatically Numbered or Alphabetized Lists

The list initialization macro for numbered lists has the form:

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

The .AL macro is used to begin sequentially numbered or alphabetized lists. If there are no arguments, the list is numbered, and text is indented by Li, initially 6 spaces from the indent in force when the .AL is called, thus leaving room for a space, two digits, a period, and two spaces before the text. Values that specify indentation must be unscaled and are treated as character positions, i.e., as the number of ens in **troff**.

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

    .nr Ls 0

specifies that no spacing will occur around any list items. The default value for Ls is 6 (which is the maximum list nesting level).

The *type* argument may be given to obtain a different type of sequencing, and its value should indicate the first element in the sequence desired, (i.e., it must be 1, A, a, I, or i). Note that the 0001 format is not permitted. If *type* is omitted or null, then 1 is assumed. If *text-indent* is non-null, it is used as the number of spaces from the current indent to the text, it is used instead of Li for this list only. If *text-indent* is null, then the value of Li will be used.

If the third argument is given, a blank line will not separate the items in the list. A blank line will occur before the first item, however.

Bullet List

The list-initialization macro for a bullet list has the form:

    .BL [text-indent] [1]

.BL begins a bullet list, in which each item is marked by a bullet (•) followed by one space. If *text-indent* is non-null, it overrides the default indentation--the amount of paragraph indentation as given in the register Pi. In the default case, the text of bullet and dash lists lines up with the first line of indented paragraphs. If a second argument is specified, no blank lines will separate the items in the list.

Dash List

The list-initialization macro for dash lists has the form:

    .DL [text-indent] [1]

.DL is identical to .BL, except that a dash is used instead of a bullet.

Marked List

The form of the list-initialization macro for a marked list is:

    .ML mark [text-indent] [1]

.ML is much like .BL and .DL, except that it requires an arbitrary mark, which may consist of more than a single character. Text is indented *text-indent* spaces if the second argument is not null; otherwise, the text is indented one more space than the width of the mark. If the third argument is specified, no blank lines will separate the items in the list. Note that the mark must not contain ordinary (paddable) spaces, because alignment of items will be lost if the right margin is justified.

Reference List

The list-initialization macro for a reference list has the form:

    .RL [text-indent] [1]

A .RL macro begins an automatically numbered list in which the numbers are enclosed by square brackets ([ ]). The *text-indent* may be supplied, as for .AL. If omitted or null, it is assumed to be 6, a convenient value for lists numbered up to 99. If the second argument is specified, no blank lines will separate the items in the list.

Variable-Item List

The list-initialization macro for a variable-item list is:

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

When a list begins with a .VL, there is effectively no current mark; it is expected that each .LI provides its own mark. This form is typically used to display definitions of terms or phrases. *Mark-indent* gives the number of spaces from the current indent to the beginning of the mark, and it defaults to 0 if omitted or null. *Text-indent* gives the distance from the current indent to the beginning of the text. If the third argument is specified, no blank lines will separate the items in the list. Here is an example of .VL usage:

```
.tr
.VL 20 2
.LI  mark1
Here is a description of mark 1;
mark 1 of the .LI line contains a tilde translated
to an unpaddable space in order to avoid extra spaces
between the mark and 1.
.LI  secondmark
This is the second mark, also using a tilde translated
to an unpaddable space.
.LI  thirdmarklongerthanindent:
This item shows the effect of a long mark; one space separates the mark
from the text.
.LI
This item has no mark because the
tilde following the .LI is translated into a space.
.LE
```

This yields:

mark1   Here is a description of mark 1; mark 1 of the .LI line
                         contains a tilde translated to an unpaddable space in order
                         to avoid extra spaces between the mark and 1.

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

thirdmarklongerthanindent: This item shows the effect of a long mark; one
                         space separates the mark from the text.

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

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

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

yields:

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

Note that the mark must not contain ordinary (paddable) spaces, because alignment of items will be lost if the right margin is justified.

List-Begin Macro and Customized Lists

The list-begin macro has the form:

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

The list-initialization macros should be adequate for most cases. However, if necessary, you may obtain more control over list layouts by using the basic list-begin macro .LB.

A *text-indent* argument gives the number of spaces that the text is to be indented from the current indent. Normally, this value is taken from the register Li for automatic lists and from the register Pi for bullet and dash lists. The combination of *mark-indent* and *pad* determines the placement of the mark. The mark is placed within an area (called "mark area") that starts mark-indent spaces to the right of the current indent, and ends where the text begins *text-indent* spaces to the right of the current indent. The *mark-indent* argument is typically 0. Within the mark area, the mark is left- justified if pad is 0. If pad is greater than 0, then n blanks are appended to the mark; the *mark-indent* value is ignored. The resulting string immediately precedes the text. That is, the mark is effectively right-justified *pad* spaces immediately to the left of the text.

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

Each nonzero value of type from 1 to 6 selects a different way of displaying the items. The following table shows the output appearance for each value of type:

Type	Appearance
1	x.
2	x)
3	(x)
4	[x]
5	<x>
6	{x}

The mark must not contain ordinary (paddable) spaces, because alignment of items will be lost if the right margin is justified.

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

There are three reasonable combinations of LI-space and LB-space. The normal case is to set LI-space to 1 and LB-space to 0, yielding one blank line before each item in the list; such a list is usually terminated with a .LE 1 to end the list with a blank line. For a more compact list, set LI-space to 0 and LB-space to 1, and, again, use .LE 1 at the end of the list. The result is a list with one blank line before and after it. If you set both LI-space and LB-space to 0, and use .LE to end the list, a list without any blank lines will result.

Displays

Displays are blocks of text that are to be kept together rather than split across pages. **MM** provides two styles of displays: a "static" (.DS) style and a "floating" (.DF) style. In the static style, the display appears in the same relative position in the output text as it does in the input text. If the display will not fit in the space remaining on a page, it will be shifted to the top of the next page. This may result in extra whitespace at the bottom of some pages. 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 precede it in the output text. A queue of floating displays is maintained so that their relative order is not disturbed.

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

Displays and footnotes can never be nested in any combination. Although lists and paragraphs are permitted, no headings (.H or .HU) can occur within displays or footnotes.

Static Displays

A static display macro has the form:

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

A static display is started by the .DS macro and terminated by the .DE macro. With no arguments, .DS will accept the lines of text exactly as they are typed (no-fill mode) and will not indent them from the prevailing left margin indentation or from the right margin. The *rindent* argument is the number of characters that the line length should be decreased, i.e., an indentation from the right margin. This number must be unscaled in **nroff** and is treated as ens. It may be scaled in **troff** or else it defaults to ems.

The *format* argument to .DS is an integer or letter used to control the left margin indentation and centering. The format argument can have the following meanings:

Code	Meaning
" "	No indent
0 or L	No indent
1 or I	Indent by standard amount
2 or C	Center each line
3 or CB	Center as a block

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

Code	Meaning
" "	-fill mode
0 or N	No-fill mode
1 or F	Fill mode

Omitted arguments are taken to be zero.

The standard indentation is taken from the Si register which is initially set at 5. Thus, by default, the text of an indented display aligns with the first line of indented paragraphs, whose indent is contained in the Pi register. Even though their initial values are the same, these two registers are independent of one another.

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

By default, a blank line is placed before and after displays. The blank lines before and after static displays can be inhibited by setting the register Ds to 0.

Floating Displays

The floating display macro has the form:

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

A floating display is started by the .DF macro and terminated by the .DE macro. The arguments have the same meanings as for .DS (see previous section on "Static Displays"), except that for floating displays, indent, no indent, and centering are always calculated with respect to the initial left margin, because the prevailing indent may change between the time when the formatter first reads the floating display and the time that the display is printed. One blank line always occurs both before and after a floating display.

You may control output positioning of floating displays through two number registers, De and Df. When a floating display is encountered by **nroff** or **troff,** it is processed and placed onto a queue of displays waiting to be output. Displays are removed from the queue and printed in the order that they were entered on the queue, which is the order that they appear in the input file. If a new floating display is encountered and the queue of displays is empty, the new display is a candidate for immediate output on the current page. Immediate output is governed by the size of the display and the setting of the Df register. The De register controls whether or not text will appear on the current page after a floating display has been produced.

The settings for the De register are as follows:

0   Default: No special action occurs.

1   A page eject will always follow the output of each floating display, so only one floating display will appear on a page and no text will follow it.

The settings for the Df register are as follows:

0   Floating displays will not be output until end of section (when using section-page numbering) or end of document.

1   Outputs the new floating display on the current page if there is room, otherwise hold it until the end of the section or document.

2   Outputs exactly one floating display from the queue at the top of a new page or column (when in two-column mode).

3   Outputs one floating display on current page if there is room. Outputs exactly one floating display at the top of a new page or column.

4   Outputs as many displays as will fit (at least one), starting at the top of a new page or column. Note that if register De is set to 1, each display will be followed by a page eject, causing a new top of page to be reached where at least one more display will be output.

5   Default. Outputs a new floating display on the current page if there is room. Outputs as many displays as will fit starting at the top of a new page or column. Note that if register De is set to 1, each display will be followed by a page eject, causing a new top of page to be reached, where at least one more display will be output.

Note: any value greater than 5 is treated as the value 5.

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

As long as the queue contains one or more displays, new displays will be automatically added to the queue, rather than be output. When a new page is started (or when at the top of the second column in two-column mode), the next display from the queue will be 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 end of a section (when using section-page numbering) or the end of a document is reached, all displays are automatically output and removed from the queue. This will occur before an .SG, .CS, or .TC macro is processed.

A display fits 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 has been used. Wide (full page width) display will never fit in the second column of a two-column document.

Tables

The table macro has the form:

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

The table start (.TS) and table end (.TE) macros allow use of the **tbl** processor. They are used to delimit the text to be examined by the **tbl** program as well as to set proper spacing around the table. The display function and the **tbl** delimiting function are independent of one another, however. In order to keep together blocks that contain any mixture of tables, equations, filled and unfilled text, and caption lines, the .TS-.TE block should be enclosed within a display (.DS-.DE), as each display is always treated as a unit. Floating tables may be enclosed inside floating displays (.DF-.DE). (For more information on displays, see the previous section on "Displays".)

The macros .TS and .TE also permit processing of tables that extend over several pages. If a table heading is needed for each page of a multipage table, use the argument H with the .TS macro (as above). Following the options and format information, the table heading is typed on as many lines as required and followed by the .TH (table header) macro. The .TH macro must occur when .TS H is used. Note that this is not a feature of **tbl**, but rather of **MM** macro definitions.

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

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

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

Equations

The equation macro has the form:

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

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

The .EQ macro takes an argument that will be used as a label for the equation. By default, the label appears at the right margin in the vertical center of the general equation. The Eq register may be set to 1 to set the label at the left margin. The equation is centered for centered displays; otherwise the equation is adjusted to the opposite margin from the label.

Figure, Table, Equation, and Exhibit Captions

The macros for captions have the form:

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

The figure title (.FG), table title (.TB), equation caption (.EC), and exhibit caption (.EX) macros are normally used inside .DS-.DE pairs to automatically number and title figures, tables, and equations. They use register Fg, Tb, Ec, and Ex, respectively. As an example, the macro:

        .FG "This is an illustration"

yields:

<div align="center">

Figure 1. This is an illustration

</div>

Instead of "Figure" TB prints "TABLE"; .EC prints "Equation", and .EX prints "Exhibit". 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 table title. The format of the numbers may be changed using the .af request of the formatter. The format of the caption may be changed from "Figure 1. Title" to "Figure 1 – Title" by setting the Of register to 1.

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; and if *flag* is 2, override replaces the number. If the -rN5 flag is given, section-figure numbering is set automatically and 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 and equations.

List of Figures, Tables, Equations, and Exhibits

Lists of Figures, Tables, Equations, and Exhibits may be obtained. They will be printed after the Table of Contents is printed if the number registers Lf, Lt, Lx, and Le are set to 1. Lf, Lt, and Lx are 1 by default; Le is 0 by default.

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

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

Footnotes

There are two macros that delimit the text of footnotes, a string used to automatically number the footnotes, and a macro that specifies the style of the footnote text. Like displays, footnotes are processed differently from the body of the text.

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

There are two macros that delimit the text of each footnote:

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

The footnote start (.FS) macro marks the beginning of the text of the footnote, and the footnote end (.FE) macro marks its end. The label on .FS, if present, will be used to mark the footnote text. Otherwise, the number retrieved from the *F will be used. Automatically numbered and user-labeled footnotes may be intermixed. If a footnote is labeled .FS the text to be footnoted must be followed by "label," rather than by *F. The text between .FS and .FE is processed in fill mode. Another .FS, a .DS, or a .DF are not permitted between the .FS and .FE macros. Automatically numbered footnotes may not be used for information, such as the title and abstract, to be placed on the cover sheet, but labeled footnotes are allowed. Similarly, only labeled footnotes may be used with tables. Here are two examples:

1. Automatically numbered footnote:

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

2. Labeled footnote:

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

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

Format of Footnote Text

The footnote format macro has the form:

    .FD [arg] [1]

Within the 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. The .FD macro is invoked to select the appropriate style. The first argument should be a number from the left column of the following table. The formatting style for each number is given by the remaining four columns. For further explanation of the first two of these columns, see the definitions of the .ad, .hy, .na, and .nh requests.

ARGUMENT	FORMATTING STYLE			
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 indent	label left-justified
5	.hy	.ad	no indent	label left-justified
6	.nh	.na	no indent	label left-justified
7	.hy	.na	no 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

If the first argument to .FD is out of range, the effect is as if .FD 0 were specified. If the first argument is omitted or null, the effect is equivalent to .FD 10 in **nroff** and to .FD 0 in **troff**; these are also the respective initial defaults.

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

    .FD " " 1

maintains the default formatting style and causes footnotes to be numbered beginning with 1 after each first-level heading.

For long footnotes that continue onto the following page, it is possible that, if hyphenation is permitted, the last line of the footnote on the current page will be hyphenated. Except for this case (which you can change by specifying an even-numbered argument to .FD), hyphenation across pages is inhibited by **MM**.

Footnotes are separated from the body of the text by a short rule. 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.

Normally, one blank line (a three-point vertical space) separates the footnotes when more than one occurs on a page. To change this spacing, set the register Fs to the desired value. For example:

    .nr Fs 2

will cause two blank lines (a six-point vertical space) to occur between footnotes.

Page 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 up to three lines of text associated with the header: every page, even page only, and odd page only. Thus the page header may have up to two lines of text: the line that occurs at the top of every page and the line for the even- or odd-numbered page. The same is true for the page footer. When not qualified by "even" or "odd", "header" and "footer" will mean those headers and footers that occur on every page. The default appearance of page headers and page footers is described here, followed by the methods for changing them.

Default Headers and Footers

By default, each page has a centered page number as the header. There is no default footer and no even/odd default headers or footers, except with section-page numbering.

In a memorandum or a released paper, the page header on the first page is automatically suppressed, if a break does not occur before .MT is called. Since they do not cause a break, the header and footer macros are permitted before the .MT macro call.

Page Header

The page header macro has the form:

.PH [arg]

For this and for the .EH, .OH, .PF, .EF, and .OF macros, the argument is of the form:

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

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

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

If "debug mode" is set using the flag -rD1 on the command line, additional information, printed at the top left of each page, is included in the default header.

Even-Page Header

The even-page header macro has the form:

.EH [arg]

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

Odd-Page Header

The odd-page header macro has the form:

.OH [arg]

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

Page Footer

The form of the page footer macro is:

    .PF [arg]

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

Even-Page Footer

The even-page footer macro has the form:

    .EF [arg]

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

Odd-Page Footer

The odd-page footer macro has the form:

    .OF [arg]

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

Footer on the First Page

By default, the footer on the first page is a blank line. If, in the input text, you specify .PF and/or .OF before the end of the first page of the document, then these lines will appear at the bottom of the first page. The header (whatever its contents) replaces the footer on the first page only if the -rN1 flag is specified on the command line.

Default Header and Footer with Section-Page Numbering

Pages can be numbered sequentially within sections. To obtain this numbering style, specify -rN3 or -rN5 on the command line. In this case, the default footer is a centered section-page number (e.g., 7-2) and the default page header is blank.

Strings and Registers in Header and Footer Macros

String and register names may be placed in the arguments to the header and footer macros. If the value of the string or register is to be computed when the respective header or footer is printed, the invocation must be escaped by four backslashes. This is because the string or register invocation is actually processed three times: as the argument to the header or footer macro; in a formatting request within the header or footer macro; and in a .tl request during header or footer processing.

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

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

This creates a right-justified header containing the word "Page" followed by the page number.

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 (i.e., top left margin of even pages and top right margin of odd pages), and "Revision 3" on the top inside margin of each page:

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

Generalized Top-of-Page Processing

This section and the next are intended only for users accustomed to writing formatter macros. During header processing, **MM** invokes two user-definable macros. One, the .TP macro, is invoked in the environment of the header. The .PX macro may be used to provide text that is to appear at the top of each page after the normal header and that may have tab stops to align it with columns of text in the body of the document.

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

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

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

For example, to obtain a page header that includes three centered lines of data, say, a document's number, issue date, and revision date, you could define .TP as follows:

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

Generalized Bottom-of-Page Processing

The bottom start macro has the form:

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

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

```
.BS
.BE
```

Top and Bottom Margins

The vertical margin macro has the form:

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

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

.VM 10 15

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

Table of Contents

The table of contents for a document is produced by invoking the table of contents (.TC) macro. The table of contents is produced at the end of the writing process because the entire document must be processed before the table of contents can be generated. The table of contents macro has the form:

.TC [slevel] [spacing] [tlevel] [tab] [head1] ... [head7]

The .TC macro generates a table of contents containing the headings that were saved for the table of contents as determined by the value of the Cl register. The arguments to .TC 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".

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

The third and fourth arguments control the placement of the page number for each heading. The page numbers can be justified at the right margin with either blanks or leader dots separating the heading text from the page number, or the page numbers can follow the heading text. For headings whose level is less than or equal to *tlevel* (default 2), 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), dots (i.e., leaders) are used; if tab is greater than 0, spaces are used. For headings whose level is greater than tlevel, the page numbers are separated from the heading text by two spaces (i.e., they are ragged right).

All additional arguments (e.g., head1, head2), if any, are horizontally centered on the page, and precede the actual table of contents itself.

If the .TC macro is invoked with at most four arguments, then the user-exit macro .TX is invoked (without arguments) before the word "CONTENTS" is printed; or the user-exit macro .TY is invoked and the word "CONTENTS" is not printed. By defining .TX or .TY and invoking .TC with at most four arguments, you can specify what needs to be done at the top of the (first) page of the table of contents.

By default, the first level headings will appear in the table of contents at the left margin. Subsequent levels will be aligned with the text of headings at the preceding level. These indentations may be changed by defining the Ci string which takes a maximum of seven arguments corresponding to the heading levels. It must be given at least as many arguments as are set by the Cl register. The arguments must be scaled. For example, with Cl = 5,

> .ds Ci .25i .5i .75i 1i 1i

or

> .ds Ci 0 2n 4n 6n 8n

Two other registers are available to modify the format of the table of contents, Oc and Cp. By default, table of contents pages will have lowercase Roman numeral page numbering. If the Oc register is set to 1, the .TC macro will not print any page number but will instead reset the P register to 1. It is your responsibility to give an appropriate page footer to place the page number. Ordinarily the same .PF used in the body of the document and exhibits will be adequate. The List of Figures and List of Tables will be produced separately unless Cp is set to 1 which causes these lists to appear on the same page as the table of contents.

References

There are two macros that delimit the text of references, a string used to automatically number the references, and an optional macro that produces reference pages within the document.

Automatic Numbering of References

Automatically numbered references may be obtained by typing *(Rf immediately after the text to be referenced. This places the next sequential reference number (in a smaller point size) enclosed in brackets a half-line above the text to be referenced.

Delimiting Reference Text

The .RS and .RF macros are used to delimit text for each reference. They have the following form:

> A line of text to be referenced.*(Rf
> .RS [string-name]
> reference text
> .RF

Subsequent References

.RS takes one argument, a "string-name". For example:

```
.RS AA
reference text
.RF
```

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

Reference Page

An automatically generated reference page is produced at the end of the document before the table of contents and the cover sheet are output. The reference page is entitled "References". This page contains the reference text (RS/RF). The user may change the reference page title by defining the RP string. For example,

```
.ds RP "New Title"
```

The optional reference page (.RP) macro may be used to produce reference pages anywhere within a document (i.e., within heading sections).

```
.RP [arg1] [arg2]
```

These arguments allow the user to control resetting of reference numbering and page skipping. The first argument with a value of 0 indicates that the reference counter is to be reset; this is the default. A value of 1 indicates that the counter will not be reset. In the second argument, a value of 0 causes a following .SK (see "Skipping Pages" later in this chapter); a value of 1 does not cause an .SK. .RP need not be used unless you want to produce reference pages elsewhere in the document.

Miscellaneous Features

In this section a number of **MM** features to control font, spacing, justification, multiple-column output and page skipping are discussed.

Bold, Italic, and Roman Fonts

Font changes are obtained with the following macros:

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

When called without arguments, .B changes the font to bold and .I changes to italic
(**troff**) or underlining (**nroff**). This condition continues until the occurrence of a .R,
when the regular Roman font is restored. Thus,

    .I
    here is some text.
    .R

yields:

here is some text.

If .B or .I is called with one argument, that argument is printed in the appropriate font
(underlined in **nroff** for .I). Then the previous font is restored (underlining is turned off
in **nroff**). If two or more arguments (maximum 6) are given to a .B or .I, the second
argument is then concatenated to the first with no intervening space (1/12-space if the
first font is italic), but is printed in the previous font; and the remaining pairs of
arguments are similarly alternated. For example:

    .I italic " text " right-justified

produces:

italic text *right*-justified

These macros alternate with the prevailing font at the time they are invoked. To
alternate specific pairs of fonts, the following macros are available:

    .IB
    .BI
    .IR
    .RI
    .RB
    .BR

Each takes a maximum of 6 arguments and alternates the arguments between the
specified fonts. Note that font changes in headings are handled separately.

Right Margin Justification

The justification macro has the form:

    .SA [arg]

The .SA macro is used to set right-margin justification for the main body of text. Two justification flags are used: "current" and "default". .SA 0 sets both flags to no justification (i.e., it acts like the .na request). .SA 1 is the inverse: it sets both flags to cause justification, just like the .ad request. However, calling .SA without an argument causes the current flag to be copied from the default flag, thus performing either an .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, the request .na can be used to ensure that justification is turned off, but .SA should be used to restore justification, rather than the .ad request. In this way, justification or lack thereof for the remainder of the text is specified by inserting .SA 0 or .SA 1 once at the beginning of the document.

SCCS Release Identification

The string .ft 1 E contains the SCCS (Source Code Control System) Release and Level of the current version of **MM.** For example, typing:

    This is version *(RE of the macros.

produces:

    This is version 15.110 of the macros.

This information is useful in analyzing suspected bugs in **MM.** The easiest way to have this number appear in your output is to specify -rD1 on the command line, which causes the string RE to be output as part of the page header.

Two-Column Output

MM can print two columns on a page:

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

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

It is possible to have full page width footnotes and displays when in two column mode, although 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 the width control (.WC) macro, which takes the following arguments:

N   Normal default mode

WF   Wide footnotes always (even in two-column mode)

-WF   Default: turns off WF (footnotes follow column mode, wide in 1C mode, narrow in 2C mode, unless FF is set)

FF   First footnote; all footnotes have the same width as the first footnote encountered for that page

-FF   Default: turns off FF (footnote style follows the settings of WF or -WF)

WD   Wide displays always (even in two-column mode)

-WD   Default: Displays follow whichever column mode is in effect when the display is encountered

For example: .WC WD FF will cause all displays to be wide and all footnotes on a page to be the same width, while .WC N will reinstate the default actions. If conflicting settings are given to .WC the last one is used. That is, .WC WF -WF has the effect of .WC -WF.

Vertical Spacing

The vertical space macro has the form:

        .SP [lines]

The .SP macro avoids the accumulation of vertical space by successive macro calls. Several .SP calls in a row produce not the sum of their arguments, but their maximum; i.e., the following produces only 3 blank lines:

        .SP 2
        .SP 3
        .SP

There are several ways of obtaining vertical spacing, all with different effects. The .sp request spaces the number of lines specified, unless no-space (.ns) mode is on, in which case the request is ignored. The .ns mode is typically set at the end of a page header in order to eliminate spacing by a .sp or .bp request that just happens to occur at the top of a page. The .ns mode can be turned off with the restore spacing (.rs) request.

Many **MM** macros utilize .SP for spacing. For example, .LE 1 immediately followed by .P produces only a single blank line between the end of the list and the following paragraph. An omitted argument defaults to one blank line (one vertical space). Negative arguments are not permitted. The argument must be unscaled but fractional amounts are permitted. Like .sp, .SP is also inhibited by the .ns request.

Skipping Pages

The skip page macro has the form:

    .SK [pages]

The .SK macro skips pages, but retains the usual header and footer processing. If *pages* is omitted, null, or 0, .SK skips to the top of the next page unless it is currently at the top of a page, in which case it does nothing. .SK n skips n pages. That is, .SK always positions the text that follows it at the top of a page, while .SK 1 always leaves one page that is blank except for the header and footer.

Forcing an Odd Page

The odd page macro has the form:

    .OP

This macro is used to ensure that the following text begins at the top of an odd-numbered page. If currently at the top of an odd page, no motion takes place. If currently on an even page, text resumes printing at the top of the next page. If currently on an odd page (but not at the top of the page) one blank page is produced, and printing resumes on the page after that.

Setting Point Size and Vertical Spacing

In **troff,** the default point size (obtained from the register S) is 10, with a vertical spacing of 12 points. The prevailing point size and vertical spacing may be changed by invoking the .S macro:

    .S [point size] [vertical spacing]

The mnemonics, D for default value, C for current value, and P for previous value, may be used for both point size and vertical spacing arguments.

Arguments may be signed or unsigned. If an argument is negative, the current value is decremented by the specified amount. If the argument is positive, the current value is incremented by the specified amount. If an argument is unsigned, it is used as the new value. .S without arguments defaults to previous (P). If the first argument is specified but the second argument (vertical spacing) is not then the default (D) value is used. The default value for vertical spacing is always 2 points greater than the current point size value selected. Footnotes are printed in a size 2 points smaller than the point size of the body, with an additional vertical spacing of 3 points between footnotes. A null (" ") argument for either the first or second argument defaults to the current (C) value.

Inserting Text Interactively

The read insertion macro has the form:

    .RD [prompt] [diversion] [string]

The read insertion macro (.RD) allows you to stop the standard output of a document and to read text from the standard input until two consecutive newlines are found. When the newlines are encountered, normal output is resumed.

.RD follows the formatting conventions already in effect. Thus, the examples below assume that the .RD is invoked in no fill mode (.nf). The first argument is a *prompt* which will be printed at the terminal. If no prompt is given, .RD signals the user with a bell on terminal output.

The second argument, a *diversion* name, allows the user to save all the entered text typed after the prompt. The third argument, a *string* name, allows the user to save for later reference the first line following the prompt. For example:

    .RD Name aa bb

produces

        Name: C. R. Jones
        16 Densmore St,
        Kensington

The diversion *aa* contains:

        C. R. Jones
        16 Densmore St,
        Kensington

The string *bb* contains *C.R. Jones.*

A newline followed by a CNTRL-D (ASCII end-of-file) also allows you to resume normal output.

Memorandum and Released Paper Styles

MM lets you specify a style for a memorandum or technical paper with a macro that controls the layout of heading information (e.g. title, author, date, etc.) on the first page or cover sheet. The information is entered in the same way for both styles; an argument indicates which style is being used. The macros used to specify paper style are described in this section.

Note that it is critical to enter the macros in the order prescribed here. If neither the memorandum nor released-paper style is desired, the macros described below should be omitted from the input text. If these macros are omitted, the first page will simply have the page header followed by the body of the document.

Title

The title macro has the form:

.TL
one or more lines of title text

The title of the memorandum or paper follows the .TL macro and is processed in fill mode. On output, the title appears after the word "subject" in the memorandum style. In the released-paper style, the title is centered and bold.

Authors

The author macro has the form:

        .AU name [initials]
        .AT [title] ...

A separate .AU macro is required for each author named.

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

        .AU "C. R. Jones" [initials] [loc] [dept] [ext] [room]
        .AT "Editor-in-chief"

In the "from" portion for the memorandum style, the author's name is followed by location and department number on one line and by room number and extension number on the next. The x for the extension is added automatically. The printing of the location, department number, extension number, and room number may be suppressed on the first page of a memorandum by setting the register Au to 0; the default value for Au is 1. Arguments 7 through 9 of the .AU macro, if present, will follow this "normal" author information in the "from" portion, each on a separate line. If your organization has a numbering scheme for memoranda, engineer's notes, etc., these numbers are printed after the author's name. This can be done by providing extra arguments to the .AU macro.

The name, initials, location, and department are also used in the Signature Block described below. The author information in the from portion, 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 released-paper style are centered below the title.

Technical Memorandum Numbers

The technical memorandum macro has the form:

    .TM [number] ...

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

    .TM 7654321 77777777

If present, this macro will be ignored in papers assigned the released-paper or external-letter styles.

Abstract

The abstract macro has the form:

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

Three styles of cover sheet are available: Technical Memorandum, Memorandum for File, and released-paper. On the cover sheet, the text of the abstract follows the author information and is preceded by the centered and underlined (italic) word "ABSTRACT".

The abstract start (.AS) and abstract end (.AE) macros bracket the abstract. The abstract is optional except that for the Memorandum for File style no cover sheet will be produced unless an abstract is given.

A combination of the first argument to .AS and the use of the .CS macro (see the section on "Cover Sheet" in this chapter) controls the production of the cover sheet. If the first argument is 2, a Memorandum for File cover sheet is generated automatically. Any other value for the first argument causes the text of the abstract to be saved until the .CS macro is invoked, then the appropriate cover sheet (either Technical Memorandum or released paper depending on the .MT type) is generated. Thus, .CS is not needed for Memorandum for File cover sheets. Notations, such as a copy to list, are allowed on Memorandum for File cover sheets. In the command sequence, the .NS and .NE macros follow the .AS 2 and .AE.

The abstract is printed with ordinary text margins. An indentation to be used for both margins can be specified as the second argument for .AS. Values that specify indentation must be unscaled and are treated as character positions, i.e., as the number of ens. Headings and displays are not permitted within an abstract.

Other Keywords

The keyword macro has the form:

    .OK [keyword] ...

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

Memorandum Types

The memorandum type macro has the form:

    .MT [type] [addressee]

The .MT macro controls the format of the top part of the first page of a memorandum or of a released paper, as well as the format of the cover sheets. Legal codes for type and the corresponding values are:

Code	Value
.MT " "	No memorandum type is printed
.MT 0	No memorandum type is printed
.MT	MEMORANDUM FOR FILE
.MT 1	MEMORANDUM FOR FILE
.MT 2	PROGRAMMER'S NOTES
.MT 3	ENGINEER'S NOTES
.MT 4	Released-paper style
.MT 5	External-letter style
.MT "string"	String

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

        .MT "Technical Note #5"

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

The second argument to .MT is used to give the name of the addressee of a letter. The name and page number will be used to replace the ordinary page header on the second and following pages of the letter. For example,

        .MT 1 "Charles Jones"

produces

        Charles Jones - 2

as the header on the second page.

This second argument may not be used if the first argument is 4 (the released-paper style).

In the external-letter style (.MT 5), only the title (without the word "subject:") and the date is printed in the upper left and right corners, respectively, on the first page. You would normally use this style with preprinted stationery that has the company name and address already printed on it.

Date and Format Changes

By default, the current date appears in the date part of a memorandum. This can be overridden by using:

        .ND new-date

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

Alternate First-Page Format

You can specify that the words "subject", "date", and "from" be omitted in the memorandum style by using the alternate format (.AF) macro. Unless you use the .AF macro, with your own company name as an argument, "Bell Laboratories" will automatically be printed as the company name on any papers which begin with .MT macros. Therefore, you will always want to use:

        .AF [company-name]

If an argument is given, it replaces "Bell Laboratories" without affecting the other headings. The .AF with no argument suppresses "Bell Laboratories" as well as the "subject", "date", and "from" headings. The use of .AF with no arguments is equivalent to the use of -rA1 on the command line, except that the latter must be used if it is necessary to change the line length and/or page offset (which default to 5.8i and 1i, respectively, for preprinted forms). The command line options -rOk and -rWk are not effective with .AF.

The only .AF option appropriate for **troff** is to specify an argument to replace "Bell Laboratories" with another name.

Released-Paper Style

The released-paper style is obtained by specifying:

    .MT 4 [1]

This results in a centered, bold title followed by centered names of authors. The location of the last author is used as the location following "Bell Laboratories" unless .AF is used to specify a different company. If the optional second argument to .MT 4 is given, then the name of each author is followed by the respective company name and location. Information necessary for the memorandum style but not for the released-paper style is ignored. The Signature Block macros and their associated lines of input are also ignored when the released-paper style is specified.

In addition to using the .AF macro to specify your company name, you can define a string with a two-character name for your address before each .AU. For example:

    .TL
    A Learned Treatise
    .AF "Getem, Inc."
    .ds XX "22 Maple Avenue, Sometown 09999"
    .AU "F. Swatter" " " XX
    .AF "Profit Associates"
    .AU "Sam P. Lename" " " " CB
    .MT 4 1

Order of Invocation of Beginning Macros

The macros described in this section must be given in the following order if they are used to define document style:

```
.ND new-date
.TL
one or more lines of text
.AF [company-name]
.AU name [initials] [loc] [dept] [ext] [room] [arg] [arg] [arg]
.AT [title] ...
.TM [number] ...
.AS [arg] [indent]
one or more lines of text
.AE
.NS [arg]
one or more lines of text
.NE
.OK [keyword] ...
.MT [type] [addressee]
```

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

Macros for the End of a Memorandum

At the end of a memorandum (but not of a released paper), the signatures of the authors
and a list of notations can be requested. The following macros and their input are
ignored if the released-paper style is selected. A signature block macro is provided in
the form:

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

.FC prints "Yours very truly" as a formal closing. It must be given before the .SG which
prints the signer's name. A different closing may be specified as an argument to .FC.
.SG prints the author name(s) after the formal closing (or the last line of text). Each
name begins at the center of the page. Three blank lines are left above each name for
the actual signature. If no argument is given, the line of reference data (e.g., location
code, department number, author's initials, and typist's initials) will not appear following
the last line.

A first argument is treated as the typist's initials, and is appended to the reference
data. A null argument prints reference data with neither the typist's initials nor the
preceding hyphen.

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

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

Copy to and Other Notations

The notation macro has the form:

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

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

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

Code	Notations
.NS " "	Copy to
.NS 0	Copy to
.NS	Copy to
.NS 1	Copy (with att.) to
.NS 2	Copy (without att.) to
.NS 3	Att.
.NS 4	Atts.
.NS 5	Enc.
.NS 6	Encs.
.NS 7	Under Separate Cover
.NS 8	Letter to
.NS 9	Memorandum to
.NS "string"	Copy (string) to

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

    .NS "with att. 1 only"

generates "Copy (with att. 1 only) to" as the notation. More than one notation may be specified before the .NE occurs, because a .NS macro terminates the preceding notation, if any.

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

Approval Signature Line

The approval signature macro has the form:

.AV "Jane Doe"

It can be used to provide a space for an approval signature next to the printed name.

Forcing a One-Page Letter

At times it is useful to get a bit more space on the page, by forcing the signature or items within notations onto the bottom of the page, so that the letter or memo is just one page in length. This can be accomplished by increasing the page length through the -rL*n* option, e.g. -rL90. This has the effect of making the formatter believe that the page is 90 lines long and therefore giving it more room than usual to place the signature or the notations. This will only work for a single-page letter or memo.

Cover Sheet

The cover sheet macro has the form:

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

The .CS macro generates a cover sheet in either the Technical Memorandum (TM) or released-paper style. All of the other information for the cover sheet is obtained from the data given before the .MT macro call. If a TM style is used, the .CS macro generates the "Cover Sheet for Technical Memorandum". The data that appears in the lower left corner of the TM cover sheet (the number of pages of text, the number of other pages, the total number of pages, the number of figures, the number of tables, and the number of references) is generated automatically. These values may be changed by supplying the appropriate arguments to the .CS macro. Any values that are omitted will be calculated automatically (0 is used for other pages). If the released-paper style is used, all arguments to .CS are ignored.

Reserved Names

If you are extending, changing, or redefining existing **MM** macros, use the legal names listed in this section. The following conventions are used in this section to describe legal names:

n	Digit
a	Lowercase letter
A	Uppercase letter
x	Any letter or digit (any alphanumeric character)
s	Special character (any nonalphanumeric character)

All other characters are literals (i.e., stand for themselves).

Note that "request", "macro", and "string" names are kept by the formatters in a single internal table, so that there must be no duplication among such names. "Number register" names are kept in a separate table.

Names Used by Formatters

These are the names of the registers and requests used by **nroff** and **troff.**

Requests
      aa (most common)
      an (currently only this name used by nroff and troff: .c2)

Registers
      aa (normal)
      .x (normal)
      .s (currently only this name used by nroff and troff: .$)
      % (page number)

Names Used by MM

These are the names of the macros, strings, and registers used by **MM.**

Macros
      AA (most common, accessible to user)
      A (less common, accessible to user)
)x (internal, constant)
      >x (internal, dynamic)

Strings
      AA (most common, accessible to user)
      A (less common, accessible to user)
      ]x (internal, usually allocated to specific functions throughout)
      }x (internal, more dynamic usage)

Registers
      Aa (most common, accessible to users)
      An (common, accessible to user)
      A (accessible, set on command line)
      :x (mostly internal, rarely accessible, usually dedicated)
      ;x (internal, dynamic, temporaries)

Names Used by eqn/neqn and tbl

The equation preprocessors, **eqn** and **neqn**, use registers and string names of the form *nn*. The table preprocessor, **tbl**, uses the following names:

    a- a+ a | nn #a ## #- #^ ^a T& TW

User-Definable Names

None of the above may be used to define your own extensions. To avoid problems, use names that consist either of a single lowercase letter, or of a lowercase letter followed by anything other than a lowercase letter. The following is a sample naming convention, where *a* can be any letter:

For macros   use a lowercase letter, followed by an uppercase letter (aA), or an uppercase letter followed by a lowercase letter (Aa).

For strings   use a, followed by a parenthesis ()), a bracket (]), or a brace (}).

For registers   use a lowercase letter followed by an uppercase letter (aA).

Sample Extension

The following is an example of how **MM** macro definitions may be extended. This sequence generates and numbers the pages of appendices:

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

After the above initialization and definition, each call of the form .aH "title" begins a new page (with the page header changed to "Appendix a -n ") and generates an unnumbered heading of "title," which, if desired, can be saved for the table of contents. Those who wish Appendix titles to be centered must, in addition, set the register Hc to 1.

Errors

When a macro discovers an error, a break occurs in processing. To avoid confusion regarding the location of the error, the formatter output buffer (which may contain some text) is printed and a short message is printed giving the name of the macro that found the error, the type of error, and the approximate line number (in the current input file) of the last processed input line. Processing terminates, unless the register D has a positive value. In the latter case, processing continues even though the output is guaranteed to be deranged from that point on.

Note that the error message is printed by writing it directly to the user's terminal. If either **tbl** or **eqn/neqn,** or both 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 results.

Disappearance of Output

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

The following command:

    grep -n "\.[EDFT][EFNQS]" files ...

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

MM Error Messages

Each **MM** error message consists of a standard part followed by a variable part. The standard part is of the form:

    ERROR:input line n

The variable part consists of a descriptive message, usually beginning with a macro name. The variable parts are listed below in alphabetical order by macro name, each with a more complete explanation:

Check TL, AU, AS, AE, MT sequence

    These macros for the beginning of a memorandum are out of sequence.

AL:bad arg:value

    The argument to the .AL macro is not one of 1, A, a, I, or i. The incorrect argument is shown as *value.*

CS:cover sheet too long

> The text of the cover sheet is too long to fit on one page. The abstract should be reduced or the indent of the abstract should be decreased.

DS:too many displays

> More than 26 floating displays are active at once, i.e., have been accumulated but not yet output.

DS:missing FE

> A display starts inside a footnote. The likely cause is the omission (or misspelling) of a .FE to end a previous footnote.

DS:missing DE

> .DS or .DF occurs within a display, i.e., a .DE has been omitted or mistyped.

DE:no DS or DF active

> .DE has been encountered but there has not been a previous .DS or .DF to match it.

FE:no FS

> .FE has been encountered with no previous .FS to match it.

FS:missing FE

> A previous .FS was not matched by a closing .FE, i.e., an attempt is being made to begin a footnote inside another one.

FS:missing DE

> A footnote start inside a display, i.e., a .DS or .DF occurs without a matching .DE.

H:bad arg:value

> The first argument to .H must be a single digit from 1 to 7, but *value* has been supplied instead.

H:missing FE

> A heading macro (.H or .HU) occurs inside a footnote.

H:missing DE

> A heading macro (.H or .HU) occurs inside a display.

H:missing arg

    .H needs at least 1 argument.

HU:missing arg

    .HU needs 1 argument.

LB:missing arg(s)

    .LB requires at least 4 arguments.

LB:too many nested lists

    Another list was started when there were already 6 active lists.

LE:mismatched

    .LE has occurred without a previous .LB or other list-initialization macro. Although this is not a fatal error, the message is issued because there almost certainly exists some problem in the preceding text.

LI:no lists active

    .LI occurs without a preceding list-initialization macro. The latter has probably been omitted, or has been separated from the .LI by an intervening .H or .HU.

ML:missing arg

    .ML requires at least 1 argument.

ND:missing arg

    .ND requires 1 argument.

SA:bad arg:value

    The argument to .SA (if any) must be either 0 or 1. The incorrect argument is shown as *value*.

SG:missing DE

    .SG occurs inside a display.

SG:missing FE

    .SG occurs inside a footnote.

SG:noauthors

.SG occurs without any previous .AU macro(s).

VL:missing arg

.VL requires at least 1 argument.

Formatter Error Messages

Most messages issued by the formatter are self-explanatory. Those error messages over which the user has some control are listed below.

Cannot do ev

Caused by setting a page width that is negative or extremely short, setting a page length that is negative or extremely short, reprocessing a macro package (e.g. performing a .so to a macro package that was requested from the command line), or requesting the -s1 option to *troff* on a document that is longer than ten pages. The term ev refers to environment parameters.

Cannot execute filename

Given by the .! request if it cannot find the filename.

Cannot open filename

Issued if one of the files in the list of files to be processed cannot be opened.

Exception word list full

Too many words have been specified in the hyphenation exception list (via .hw requests).

Line overflow

The output line being generated was too long for the formatter's line buffer. The excess was discarded. See the "Word overflow" message below.

Nonexistent font type

A request has been made to mount an unknown font.

Nonexistent macro file

The requested macro package does not exist.

Nonexistent terminal type

    The terminal options refers to an unknown terminal type.

Out of temp file space

    Additional temporary space for macro definitions, diversions, etc. cannot be
    allocated. This message often occurs because of unclosed diversions (missing .FE
    or .DE), unclosed macro definitions (e.g., missing ".."), or a huge table of contents.

Too many page numbers

    The list of pages specified to the formatter -o option is too long.

Too many string/macro names

    The pool of string and macro names is full. Unneeded strings and macros can be
    deleted using the .rm request.

Too many number registers

    The pool of number register names is full. Unneeded registers can be deleted by
    using the .rr request.

Word overflow

    A word being generated exceeded the formatter's word buffer. The excess
    characters were discarded. A likely cause for this and for the "Line overflow"
    message above are very long lines or words generated through the misuse of \c or
    of the .cu request, or very long equations produced by **eqn** or **neqn.**

Summary of Macros, Strings, and Number Registers

The following is an alphabetical list of macro names used by **MM**. The first line of each item gives the name of the macro and a brief description. The second line shows the form in which the macro is called. Macros marked with an asterisk are not, in general, invoked directly by the user. They are "user exits" called from inside header, footer, or other macros.

1C   One-column processing
        .1C

2C   Two-column processing
        .2C

AE   Abstract end
        .AE

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

AL   Automatically-incremented list start
        .AL [type] [text-indent] [1]

AS   Abstract start
        .AS [arg] [indent]

AT   Author's title
        .AT [title] ...

AU   Author information
        .AU name [initials] [loc] [dept] [ext] [room] [arg] [arg] [arg]

AV   Approval signature
        .AV [name]

B   Bold
        .B [bold-arg] [previous-font-arg] [bold] [prev] [bold] [prev]

BE   Bottom end
        .BE

BI   Bold/Italic
        .BI [bold-arg] [italic-arg] [bold] [italic] [bold] [italic]

BL   Bullet list start
        .BL [text-indent] [1]

BR   Bold/Roman
      .BR [bold-arg] [Roman-arg] [bold] [Roman] [bold] [Roman]

BS   Bottom start
      .BS

CS   Cover sheet
      .CS [pages] [other] [total] [figs] [tbls] [refs]

DE   Display end
      .DE

DF   Display floating start
      .DF [format] [fill] [right-indent]

DL   Dash list start
      .DL [text-indent] [1]

DS   Display static start
      .DS [format] [fill] [right-indent]

EC   Equation caption
      .EC [title] [override] [flag]

EF   Even-page footer
      .EF [arg]

EH   Even-page header
      .EH [arg]

EN   End equation display
      .EN

EQ   Equation display start
      .EQ [label]

EX   Exhibit caption
      .EX [title] [override] [flag]

FC   Formal closing
      .FC [closing]

FD   Footnote default format
      .FD [arg] [1]

FE   Footnote end
      .FE

FG   Figure title
      .FG [title] [override] [flag]

FS   Footnote start
.FS [label]

H   Heading--numbered
.H level [heading-text] [heading-suffix]

HC   Hyphenation character
.HC [hyphenation-indicator]

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

HU   Heading--unnumbered
.HU heading-text

HX   Heading user exit X (before printing heading)
.HX dlevel rlevel heading-text

HY   Heading user exit Y (before printing heading)
.HY dlevel rlevel heading-text

HZ   Heading user exit Z (after printing heading)
.HZ dlevel rlevel heading-text

I   Italic (underline in **nroff**)
.I [italic-arg] [previous-font-arg] [italic] [prev] [italic] [prev]

IB   Italic/Bold
.IB [italic-arg] [bold-arg] [italic] [bold] [italic] [bold]

IR   Italic/Roman
.IR [italic-arg] [Roman-arg] [italic] [Roman] [italic] [Roman]

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

LC   List-status clear
.LC [list-level]

LE   List end
.LE [1]

LI   List item
.LI [mark] [1]

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

MT   Memorandum type
      .MT [type] [addressee]
      or .MT [4] [1]

ND   New date
      .ND new-date

NE   Notation end
      .NE

NS   Notation start
      .NS [arg]

nP   Double-line indented paragraphs
      .nP

OF   Odd-page footer
      .OF [arg]

OH   Odd-page header
      .OH [arg]

OK   Other keywords for TM cover sheet
      .OK [keyword] ...

OP   Odd page
      .OP

P   Paragraph
      .P [type]

PF   Page footer
      .PF [arg]

PH   Page header
      .PH [arg]

PX   Page-header user exit
      .PX

R   Return to regular (Roman) font (end underlining in **nroff**)
      .R

RB   Roman/Bold
      .RB [Roman-arg] [bold-arg] [Roman] [bold] [Roman] [bold]

RD   Read insertion from terminal
      .RD [prompt] [diversion] [string]

RF   Reference end
       .RF

RI   Roman/Italic
       .RI [Roman-arg] [italic-arg] [Roman] [italic] [Roman] [italic]

RL   Reference list start
       .RL [text-indent] [1]

RP   Produce reference page
       .RP [arg] [arg]

RS   Reference start
       .RS [string-name]

S   Set **troff** point size and vertical spacing
       .S [size] [spacing]

SA   Set adjustment (right margin justification) default
       .SA [arg]

SG   Signature line
       .SG [arg] [1]

SK   Skip pages
       .SK [pages]

SP   Space--vertically
       .SP [lines]

TB   Table title
       .TB [title] [override] [flag]

TC   Table of contents
       .TC [slevel] [spacing] [tlevel] [tab] [head1] [head2] [head3] [head4] [head5]

TE   Table end
       .TE

TH   Table header
       .TH [N]

TL   Title of memorandum
       .TL [charging-case] [filing-case]

TM   Technical Memorandum number(s)
       .TM [number] ...

TP   Top-of-page macro
       .TP

TS   Table start
      .TS [H]

TX   Table-of-contents user exit
      .TX

TY   Table-of-contents user exit (suppresses "CONTENTS")
      .TY

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

VM   Vertical margins
      .VM [top] [bottom]

WC   Width control
      .WC [format]

Strings

The following is an alphabetic list of string names used by **MM**, giving for each a brief description and an initial default value.

Ci   Contents indent up to seven arguments for heading levels.

F   Footnote numberer.
      In **nroff:** \u\\n+(:p\d
      In **troff:** \v'-.4m'\s-3\\n+9:p\s0\v'.4m'

DT   Date. The current date, unless overridden.

EM   Em dash string. Used by both **nroff** and **troff**

HF   Heading font list, up to seven codes for heading levels 1 through 7
      3 3 2 2 2 2 2 (levels 1 and 2 bold, 3-7 underlined in **nroff,** italic in **troff**)

HP   Heading point size list, up to seven codes for heading levels 1 through 7

Le   Title for LIST OF EQUATIONS

Lf   Title for LIST OF FIGURES

Lt   Title for LIST OF TABLES

Lx   Title for LIST OF EXHIBITS

RE   SCCS Release and **MM**
      Release Level

Rf   Reference numberer

Rp   Title for References

Tm   Trademark string places the letters "TM" half a line above the text that it follows

Number Registers

This section provides an alphabetical list of register names, giving for each a brief description, initial (default) value, and the legal range of values (where [m:n] means values from m to n inclusive).

Any register having a single-character name can be set from the command line. An asterisk attached to a register name indicates that that register can be set only from the command line or before the **MM** macro definitions are read by the formatter.

A   Handles preprinted forms
      0, [0:2]

Au   Inhibits printing of author's location, department, room, and extension in the from portion of a memorandum
      1, [0:1]

C   Copy type (Original, DRAFT, etc.)
      0 (Original), [0:4]

Cl   Contents level (i.e., level of headings saved for table of contents)
      2, [0:7]

Cp   Placement of List of Figures, etc.
      1 (on separate pages), [0:1]

D   Debug flag
      0, [0:1]

De   Display eject register for floating displays
      0, [0:1]

Df   Display format register for floating displays
      5, [0:5]

Ds   Static display pre- and post-space
      1, [0:1]

Ec   Equation counter, used by .EC macro
        0, [0:?], incremented by 1 for each .EC call.

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

Eq   Equation label placement
        0 (right-adjusted), [0:1]

Ex   Exhibit counter, used by .EX macro
        0, [0:?], incremented by 1 for each .EX call.

Fg   Figure counter, used by .FG macro
        0, [0:?], incremented by 1 for each .FG call.

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

H1-H7
        Heading counters for levels 1-7
        0, [0:?], incremented by .H of corresponding level or .HU if at level given by
        register Hu. H2-H7 are reset to 0 by any heading at a lower-numbered level.

Hb   Heading break level (after .H and .HU)
        2, [0:7]

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

Hi   Heading temporary indent (after .H and .HU)
        1 (indent as paragraph), [0:2]

Hs   Heading space level (after .H and .HU)
        2 (space only after .H 1 and .H 2), [0:7]

Ht   Heading type (for .H: single or concatenated numbers)
        0 (concatenated numbers: 1.1.1, etc.), [0:1]

Hu   Heading level (for unnumbered heading .HU)
        2 (.HU at the same level as .H 2), [0:7]

Hy   Hyphenation control for body of document
        0 (automatic hyphenation off), [0:1]

L   Length of page
        66, [20:?] (11i, [2i:?] in **troff** these values must be scaled.

Le   List of Equations
        0 (list not produced) [0:1]

Lf   List of Figures
        1 (list produced) [0:1]

Li   List indent
        6, [0:?]

Ls   List spacing between items by level
        5 (spacing between all levels)

Lt   List of Tables
        1 (list produced) [0:1]

Lx   List of Exhibits
        1 (list produced) [0:1]

N   Numbering style
        0, [0:5]

Np   Numbering style for paragraphs
        0 (unnumbered) [0:1]

O   Offset of page
        .75i, [0:?] (0.5i, [0i:?] in **troff**

Oc   Table of Contents page numbering style
        0 (lowercase Roman), [0:1]

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

P   Page number, managed by **MM.**
        0, [0:?]

Pi   Paragraph indent
        5, [0:?]

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

Pt   Paragraph type
        0 (paragraphs always left-justified), [0:2]

S   Point size
        10, [6:36]

Si   Standard indent for displays
        5, [0:?]

T   Type of **nroff** output device
        0, [0:2]

Tb   Table counter
      0, [0:?], incremented by 1 for each .TB call.

U   Underlining style for .H and .HU
      0 (continuous underline when possible), [0:1]

W   Width of page (line and title length)
      6i, [10:1365] (6i, [2i:7.54i] in **troff**

Introduction

Nroff and **troff** are the XENIX text formatting programs for producing high-quality printed output on the lineprinter and phototypesetter, respectively. Commands in the two formatting programs **nroff** and **troff** are identical, although those specifications which are impossible to achieve on a lineprinter--like changes in point size, font, or variable spacing--are either approximated or ignored by **nroff**. The output of **nroff** and **troff** may look dramatically different, but this is largely the result of the limitations of conventional lineprinters. In this chapter, the two programs will be treated together; the names **nroff** and **troff** are used synonymously. Commands not recognized by **nroff** or which result in significantly different output will be noted.

Wherever possible, you should avoid using **nroff** or **troff** directly. In many ways, **nroff** and **troff** resemble computer assembly languages: they are powerful and flexible, but they require that many operations must be specified at a level of detail and complexity too difficult for most people to use effectively. That is why it is suggested that you use the **MM** macro package instead. If you must deal with specialized text, you can use the **eqn** macros for typesetting mathematics and the **tbl** program for producing complex tables. **Eqn** and **tbl** are discussed in Chapters 10 and 11 of this manual.

For producing running text, whether or not it contains mathematics or tables, you will ordinarily want to use the **MM** macro package, described in Chapter 3, "Using the MM Macros" and Chapter 4, "MM Reference".

All these macro packages offer the capability of meeting most formatting requirements. You may find you have little or no need to use **nroff/troff** directly. The macros define formatting rules and operations for specific styles of documents. The definitions are concise: in most cases two-letter commands. In those cases where an existing macro will not do the job, the solution is not to write an entirely new set of **nroff/troff** instructions from scratch, but to make small adaptations to macros you are already using.

This chapter is meant to introduce you to the formatting possibilities of **nroff/troff**. It does not discuss every command or operation in detail. The emphasis is on demonstrating relatively simple and commonly used specifications, with examples of some of the variations you may need to create.

The following topics are introduced in this tutorial:

●   Specifying point size, fonts, and special characters

●   Determining line spacing, line lengths, indents, and tabs

●   Using string definitions and macros

- Specifying title and pagination styles

- Specifying conditionals, environments, and diversions

Inserting Commands

To use **nroff** or **troff** you intersperse formatting commands with the actual text you want printed, just as you did with **MM** commands described in the last chapter. You will notice that **nroff** and **troff** commands are in lowercase, so you will not confuse them with the **MM** macros. Most **nroff** and **troff** commands are placed on a line separate from the text itself, beginning with a period, one command per line. For example, if you had a file that contained the following lines:

        Some text.
        .ps 14
        Some more text.

the .ps command would instruct **troff** to change the point size, that is, the size of the letters being printed, to 14 point (one point is 1/72-inch). Your output would look like this:

        Some text. Some more text.

If you were to use **nroff** to output this same file to the lineprinter, **nroff** would ignore the .ps command and you would see no difference in the size of your letters.

Some **nroff/troff** commands do occur in the middle of a line. To produce

        This **line** contains *font* and point size changes.

you have to type

        This \fBline\fR contains \fIfont and \s+2point size\s-2 changes.

The backslash character "\" is used to introduce **nroff/troff** commands and special characters within a line of text.

Point Sizes and Line Spacing

As we just saw, point size and vertical spacing are not normally controllable in **nroff** (lineprinter) output. In **troff**, the command .ps sets the point size. One point is 1/72-inch, so 6-point characters are at most 1/12-inch high, and 36-point characters are 1/2-inch. There are 15 point sizes, including:

8 point: In Xanadu did Kubhla Khan...

10 point: In Xanadu did Kubhla Khan...

11 point: In Xanadu did Kubhla Khan...

12 point: In Xanadu did Kubhla Khan...

14 point: In Xanadu did Kubhla Khan...

18 point: In Xanadu did Kubhla Khan...

24 point: In Xanadu did Kubhla Khan..

If the number after .ps is not one of the legal sizes, it is rounded up to the next valid value, to a maximum of 36. If no number follows .ps, **troff** reverts to its previous size. **Troff** begins with a default point size of 10.

Point size can also be changed in the middle of a line or even a word with the in-line command "\s". To produce

The XENIX system is derived from the UNIX system.

type

The \s14XENIX\s10 system is derived from the \s14UNIX\s10 system.

The \s should be followed by a legal point size. A \s0 causes the size to revert to its previous value. An \s1011 means "size 10, followed by an 11".

Relative size changes are possible. The following

The \s+2XENIX\s-2 system

increases the point size by two points, then restores it. The amount of the relative change is limited to a single digit.

Another feature to consider 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 command to control vertical spacing is .vs. For running text, it is usually best to set the vertical spacing about 20% bigger than the point size.

For example, to use what typesetters call "9 on 11", that is, a point size of 9 with a vertical spacing of 11, you would insert the following commands:

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

If you do not specify a point size or vertical spacing, **troff** automatically uses 10 on 12.

Point size and vertical spacing make a substantial difference in the amount of text per square inch. (This is 12 on 14.)

Point size and vertical spacing make a substantial difference in the amount of text per square inch. For example, 10 on 12 uses about twice as much space as 7 on 8. This is 8 on 9, which packs a lot of words per line.

When you use the commands .ps and .vs without numbers, **troff** reverts to the previous size and vertical spacing.

The .sp command can be used to get vertical space. Without a number, it gives you one blank line (one unit of whatever .vs has been set to). The .sp can be followed by a unit specification:

```
.sp 2i
```

means "two inches of vertical space". The command:

```
.sp 2p
```

means "two points of vertical space". The command:

```
.sp 2
```

means "two vertical spaces" of whatever size .vs is set to. Be careful to specify the correct unit of space.

Troff also understands decimal fractions in most commands, so

```
.sp 1.5i
```

is a space of 1.5 inches. Scaling (designating a unit of measure such as inches, points, or picas) can also be used after .vs to define line spacing, and in fact after most commands that deal with physical dimensions.

Fonts and Special Characters

The phototypesetter is limited to four different fonts at any one time. Normally three fonts (Roman, italic and bold) and one collection of special characters are permanently mounted. What these fonts will actually look like depends on your own typesetting equipment. Here are the Roman, italic, and bold character sets:

abcdefghijklmnopqrstuvwxyz 0123456789
ABCDEFGHIJKLMNOPQRSTUVWXYZ
abcdefghijklmnopqrstuvwxyz 0123456789
ABCDEFGHIJKLMNOPQRSTUVWXYZ
abcdefghijklmnopqrstuvwxyz 0123456789
ABCDEFGHIJKLMNOPQRSTUVWXYZ

Troff prints in Roman by default, unless instructed otherwise. To switch into bold, use the .ft (font) command

    .ft B

and for italics,

    .ft I

To return to roman, use .ft R; to return to the previous font, whatever it was, use either .ft P or just .ft. The underline command .ul causes the next input line to print in italics. The .ul can be followed by a count to indicate that more than one line is to be italicized.

Fonts can also be changed within a line or word with the in-line command "\f". The words

    bold*face* text

are produced with

    \fBbold\fIface\fR text

There are other fonts available besides the standard set, although only four can be mounted at any given time. The command .fp tells **troff** what fonts are physically mounted on the typesetter:

    .fp 3 H

says that the Helvetica font is mounted on position 3. Appropriate .fp commands should appear at the beginning of your document if you do not use the standard fonts.

It is possible to print a document by using font numbers instead of names. For example, \f3 and .ft 3 mean "whatever font is mounted at position 3". Normal settings are Roman font on 1, italic on 2, bold on 3, and special on 4. An approximation of bold font can also be created by overstriking letters with a slight offset. This is done with the command .bd.

Special characters have four-character names beginning with "\(", and they may be inserted anywhere. In particular, Greek letters are all of the form "\(*-", where "-" is an uppercase or lowercase Roman letter similar to the Greek. To get

$\Sigma(\alpha X \beta) \to \infty$

in **troff** we have to type

    \(*S(\(*a\(mu\(*b) \(\(-> \(if

which is a series of special characters:

    \(*S   Σ
    ((
    \(*a   α
    \(mu   X
    \(*b   β
)   )
    \(->   →
    \(if   ∞

You could also use the mathematical typesetting program **eqn** to achieve the same effect:

    SIGMA (alpha times beta) -> inf

Whether you choose to use **eqn** or the **troff** special character set should depend on how often you use Greek or other special characters.

Nroff and **troff** treat each four-character name as a single character. Some characters are automatically translated into others: grave and acute accents (apostrophes) become open and close single quotation marks (' '); the combination of single quotation marks is generally preferable to the double quotation mark character ("). A typed minus sign becomes a hyphen -. To print an explicit minus sign, use "\-". To get a backslash printed literally, use "\e".

Indents and Line Lengths

Troff starts with a line length of 6.5 inches. To reset the line length, use the .ll (line length) command, as in

    .ll 6i

to indicate a line length of 6 inches. The length can be specified in the same ways as the space (.sp) command, in inches, fractions of inches, or points.

The maximum line length provided by the typesetter is 7.5 inches. To use the full width, however, you will have to reset the default physical left margin, which is normally slightly less than one inch from the left edge of the paper. This is done with the page offset (.po) command:

    .po 0

This sets the offset as far to the left as it will go.

The indent (.in) command causes the left margin to be indented by a specified amount from the page offset. If we use .in to move the left margin in, and .ll to move the right margin to the left, we can make offset blocks of text. For example,

```
.in 0.6i
.ll -0.6i
text to be set into a block
.ll +0.6i
.in -0.6i
```

will create a block that looks like this:

> Pater noster qui est in caelis sanctificetur nomen tuum; adveniat regnum tuum; fiat voluntas tua, sicut in caelo, et in terra...
> Amen.

Notice the use of + and - to specify the amount of change. These change the previous setting by the specified amount, rather than just overriding it. The distinction is quite important: .ll +1i makes lines one inch longer than current setting: .ll 1i makes them one inch long. If no argument is specified with .in, .ll, and .po, **troff** reverts to the previous value.

To indent a single line, use the temporary indent (.ti) command. The default unit for .ti, as for most horizontally oriented commands such as .ll, .in, .po, is an em. An em is roughly the width of the letter m in the current point size. Although inches may seem a more intuitive measure to nontypesetters, ems are a measure of 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 ems for all dimensions. Ems can be specified in the same way as points or inches:

```
.ti 2.5m
```

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

```
.ti -0.3i
```

causes the next line to be moved back three-tenths of an inch. You can make a decorative initial capital, indent a whole paragraph, and move the initial letter back with a .ti command:

P ater noster qui est in caelis sanctificetur nomen tuum;
 adveniat regunm tuum;
fiat voluntas tua, sicut in caelo, et in terra. . . . Amen.

This is achieved with the following:

```
.ll -0.3i
.fi
.in +3i
.ti -0.3i
```

The P is made bigger with a "\s36P\s0". It also has been moved down from its normal position with a local motion, as described in the section on "Drawing Lines and Characters" later in this chapter.

Tabs

Tabs can be used to produce output in columns, or to set the horizontal position of output. Typically, tabs are used only in unfilled text. Tab stops are set by default every 1/2-inch from the current indent, but can be changed with the .ta command. To set stops every inch, for example, use:

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

The stops are only left-justified, as they are on a typewriter, so lining up columns of right-justified numbers can be painful. If you have many numbers, or if you need more complicated table layout, don't attempt to use **nroff** or **troff** commands. Use the **tbl** program instead. (See Chapter 7, "Formatting Tables".)

For a handful of numeric columns, you can precede every number by enough blanks to make it line up when typed:

```
.nf
.ta 1i 2i 3i
  1   tab   2   tab   3
 40   tab  50   tab  60
700   tab 800   tab 900
.fi
```

Then change each leading blank into the string "\0". This is a character that does not print, but that has the same width as a digit. When printed, this will produce

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

It is also possible to fill up tabbed-over space with a character other than a space by setting the "tab replacement character" with the tab character (.tc) command:

```
.ta 1.5i 2.5i
.tc \(ru
Name tab Age tab
```

produces

Name_____Age_____

To reset the tab replacement character to a blank, use .tc with no argument. Lines can also be drawn with the \l command, described below.

Drawing Lines and Characters

Troff provides a way to place characters of any size at any place, as in the examples Area = πr^2 and the big P in the Pater noster (see section on "Indents and Line Lengths" in this chapter). Commands can be used to draw special characters or to give your output a particular appearance. Most of these commands are reasonably straightforward, but look rather complicated.

For example, without **eqn,** subscripts and superscripts are most easily done with the half-line local motions \u and \d. To go back up the page half a point-size, insert a \u at the desired place; to go down, insert a \d. Thus

        Area = \(*pr\u2\d

produces

        Area = πr^2

To make the 2 smaller, bracket it with

        \s-2...\s0

Since \u and \d are relative to the current point size, be sure to put them either both inside or both outside the size changes, or the results will be unbalanced.

If the space given by \u and \d doesn't look right, the \v command can be used to request an arbitrary amount of vertical motion. The in-line command

        \v'(amount)'

causes motion up or down the page by the specified amount. For example, to move the P in Pater, the following is required:

        .ta 1i
        .in +0.6i   \"move paragraph in
        .ll -0.3i   \"shorten lines
        .ti -0.3i   \"move P back
        \v'1'\s36P\s0\v'\-1'ater noster qui est
        in caelis ...

The backslash \" is a **troff** command that causes the rest of the line to be ignored. It is useful for adding comments to the macro definition.

A minus sign after "\v" causes upward motion, while no sign or a plus sign causes downward motion. Thus "\v' -1' " causes an upward vertical motion of one line space.

There are many other ways to specify the amount of motion:

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

and so on are all legal. Notice that the specifiers, i for inches, p for points or m for ems, go inside the quotation marks. Any character can be used in place of the quotation marks, as well as in any **troff** commands described in this section.

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

Arbitrary horizontal motions are also available: \h is quite analogous to \v, except that its default scale is ems instead of line spaces. The specification \h'-0.1i' causes a backwards motion of a 1/10-inch.

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

        \w'thing'

is a number equal to the width of thing in machine units (1/432-inch). All **troff** computations are actually done in these units. To move horizontally the width of an x, you can use:

        \h'\w'x'u'

As we mentioned above, the default scale factor for all horizontal dimensions is m for ems, so here u for machine units must be specified, or the motion produced will be far too large. Nested quotation marks are acceptable to **troff**; be careful to supply the right number.

There are also several special-purpose **troff** commands for local motion. We have already seen \0, which is an unpaddable whitespace of the same width as a digit. Unpaddable means that it will never be widened or split across a line by line justification and filling. There is also \(space), which is an unpaddable character the width of a space, \|, which is half that width, \^, which is one quarter of the width of a space, and \&, which has zero width. This last one is useful, for example, when entering a text line which would otherwise begin with a dot (.).

The command "\o", used like

        \o'set of characters'

causes up to 9 characters to be overstruck, centered on the widest. This can be used for accents, as in:

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

which makes:

        système téléphonique

The accents are treated by **troff** as single characters.

You can make your own overstrikes with another special convention, \z, the zero-motion command, which suppresses the normal horizontal motion after printing the single character x, so another character can be laid on top of it. Although sizes can be changed within \o, it centers the characters on the widest, and there can be no horizontal or vertical motions, so \z may be the only way to get what you want.

You can create rather ornate overstrikes with the bracketing function \b, which piles up characters vertically, centered on the current baseline. Thus you can get big brackets by constructing them with piled-up smaller pieces:

        $$\left\{\left[\ x\ \right]\right\}$$

by typing in this:

        \b'\(lt\(lk\(lb'\b'\(lc\(lf x \b'\(rc\(rf\b'\(rt\(rk\(rb'

Troff also provides a convenient facility for drawing horizontal and vertical lines of arbitrary length with arbitrary characters. \l' 1i' draws a line one inch long, like this: _____. The length can be followed by the character to use if the _ isn't appropriate. For example, \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.

Strings

Obviously if a paper contains a large number of occurrences of an acute accent over a letter e, typing \o"e\' " for each occurrence would be a great nuisance. Fortunately, **nroff** and **troff** provide a facility for storing any string of text in a string definition. Strings are among the **nroff** and **troff** mechanisms that allow you to type a document with less effort and organize it so that extensive format changes can be made with few editing changes. Strings are defined with the define (.ds) command. Thereafter, whenever you need to use the string, you can replace it with the shorthand you have defined. For example, the line:

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

defines the string e to have the value é.

String names may be either one or two characters long. To distinguish them from normal text, single-character strings must be preceded by "*" and double-character strings by "*(". Thus to use the definition of the string e as above, we can say t*el*ephone. If a string must begin with blanks, define it by using a double quotation mark to signal the beginning of the definition. For example,

```
.ds xx "    text
```

defines the string "xx" as the word "text" preceded by several blanks. There is no trailing quote; the end of the line terminates the string.

A string may actually be several lines long; if **troff** encounters a \ at the end of any line, it is thrown away and the next line added to the current one. So you can make a long string simply by ending each line but the last with a backslash:

```
.ds xx this is a very long string \
continuing on the next line \
and on to the next
```

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

Macros

In its simplest form, a macro is just a shorthand notation--somewhat like a string. For example, suppose we want every paragraph in a document to start with a space and a temporary indent of two ems:

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

To save typing, we could translate these commands into one macro:

```
.P
```

which **troff** would interpret exactly as

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

If you first define it with the .de command, the macro .P can replace the longer specification:

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

The first line names the macro, in this case .P for paragraph; it is in uppercase to avoid conflict with any existing **nroff** or **troff** command. The last line marks the end of the definition. In between is the text, which is simply inserted whenever **troff** sees the command or macro call .P. A macro can contain any mixture of text and formatting commands. The definition of .P naturally has to precede its first use. Names are restricted to one or two characters.

Using macros for commonly occurring sequences of commands not only saves typing, but it makes later changes much easier. Suppose we 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, we need only change the definition of .P to something like

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

and the change takes effect everywhere the .P macro is invoked.

As another example of a macro definition, 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 we can surround text with the commands .BS and .BE to create indented blocks. Uses of .BS and .BE can be nested to get blocks within blocks. To change the indent, it is only necessary to change the definitions of .BS and .BE, not every occurrence of the indent in the entire document.

The macro package **MM**, as well as the two specialized macro packages, **tbl** and **eqn**, are simply very large collections of macro definitions which replace more cumbersome arrays of **nroff** and **troff** commands. One thing to keep in mind when you consider defining a new macro, is that unless you are doing something quite unusual, an **MM** macro probably already exists for that purpose. So check your documentation carefully before reinventing the wheel.

Titles, Pages and Numbering

None of the features described in this section are automatic. You may wish to copy these specifications literally until you feel more comfortable with these commands. For example, suppose you want to have a title at the top of each page. You have to give the actual title, along with instructions about when to print it, and directions for its appearance. First, a new page (.NP) macro can be created to process titles and the like at the end of one page and the beginning of the next:

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

To start at the top of a page, a begin page (.bp) command should be included, which causes a skip to the top of the next page. Then we space down half an inch, use the title (.tl) command to print the title and space another 0.3 inches.

To ask for .NP at the bottom of each page, we need to specify that the processing for a new page should start when the text is within an inch of the bottom of the page. This is done with a when (.wh) command:

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

(Note that no dot is used before NP; this is simply the name of a macro, not a macro call.) The minus sign means "measure up from the bottom of the page," so -1i means one inch from the bottom.

The .wh command appears in the input outside the definition of .NP; typically the input would be

```
.de NP
macro defined here
..
.wh -1i NP
```

As text is actually being output, **nroff/troff** keeps track of its vertical position on the page, and after a line is printed within one inch of the bottom, the .NP macro is activated. The .NP macro causes a skip to the top of the next page, then prints the title with the appropriate margins. All the input text collected but not yet printed is flushed out as soon as possible, and the next input line is guaranteed to start a new line of output; a break is caused in the middle of the current output line when a new page is started. The leftover part of that line is printed at the top of the page, followed by the next input line on a new output line. Using \(fm instead of dot (.) for a command tells **nroff** and **troff** that no break is to take place; the output line currently being filled should not be forced out before the space or new page. For example, \(fmbp and \(fmsp are used here instead of .bp and .sp.

The list of commands that cause a break is short:

    .bp .br .ce .fi .nf .sp .in .ti

All others cause no break, regardless of whether you use a period (.) or a '. If you really need a break, add a .br command at the appropriate place.

If you change fonts or point sizes frequently, you may find that if you cross a page boundary in an unexpected font or size, your titles come out in that size and font instead of what you intended. Furthermore, the length of a title is independent of the current line length, so titles will come out at the default length of 6.5 inches unless you change it, which is done with the .lt command. There are several ways to correct point sizes and fonts in titles. The simplest way is to change .NP to set the proper size and font for the title, then restore the previous values, like this:

    .ta .8i
    .de NP
    ' bp
    'sp 0.5i
    .ft R   \" set title font to Roman
    .ps 10   \" and size to 10 point
    .lt 6i   \" and length to 6 inches
    .tl 'left'center'right'
    .ps   \" revert to previous size
    .ft P   \" and to previous font
    'sp 0.3i

This version of .NP does not work if the fields in the .tl command contain size or font changes.

To get a footer at the bottom of a page, you can modify .NP so it does some processing before the 'bp command, or split the job into a footer macro invoked at the bottom margin and a header macro invoked at the top of the page.

Output page numbers are computed automatically starting at 1, but no numbers are printed unless you ask for them. To get page numbers printed, include the character "%" in the .tl line at the position where you want the number to appear. For example

    .tl "- % -"

centers the page number inside hyphens. You can set the page number at any time with either .bp n, which immediately starts a new page numbered n, or with .pn n, which sets the page number for the next page but doesn't cause a skip to the new page.

Number Registers and Arithmetic

Troff uses number registers for doing arithmetic and defining and using variables. Number registers, like strings and macros, are useful for setting up a document so it is

easy to change later, as well as for doing any sort of arithmetic computation. Like strings, number registers have one- or two-character names. They are set by the .nr command, and are referenced by \nx (one-character name) or \n(xy (two-character name).

There are quite a few pre-defined number registers maintained by **troff,** among them % for the current page number, .nl for the current vertical position on the page; .dy, .mo and .yr for the current day, month and year; and .s and .f for the current point size and font. Any of these can be used in computations like any other register, but some, like .s and .f, cannot be arbitrarily changed with an .nr command.

In **MM,** most significant parameters are defined in terms of the values of a handful of number registers. These include the point size for text, the vertical spacing, and the line and title lengths. To set the point size and vertical spacing for the following paragraphs, for example, you could say

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

This would set the point size to 9 and the vertical spacing to 11 points. The paragraph macro .P is defined as follows:

```
.ta 1i
.de.P
.ps \\n(PS     \" reset size
.vs \\n(VSp    \" spacing
.ft R          \" font
.sp 0.5v       \" half a line
.ti +3m
```

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

Two backslashes are required to quote a quote. That is, when **nroff** or **troff** originally read the macro definition, they peel off one backslash to see what is coming next. To ensure that another is left in the definition when the macro is actually used, we have to put two backslashes in the definition. If only one backslash is used, point size and vertical spacing will be frozen at the time the macro is defined, not when it is used.

Protection with extra backslashes is only needed for \n, *, \$, and \ itself. Commands like \s, \f, \h, \v, and so on do not need an extra backslash, since they are converted by **nroff** and **troff** to an internal code when they are read.

Arithmetic expressions can appear anywhere that a number is expected. For example,

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

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

There are a few things to consider in using number register arithmetic. First, number registers hold only integers. **Nroff/troff** arithmetic uses truncating integer division. Second, in the absence of parentheses, evaluation is done left-to-right without any operator precedence, including relational operators. Thus

    7*-4+3/13

becomes "-1". Number registers can occur anywhere in an expression, and so can scale indicators like p, i, m, and so on. Although integer division causes truncation, each number and its scale indicator is converted to machine units (1/432-inch) before any arithmetic is done, so 1i/2u evaluates to 0.5i correctly.

The scale indicator u (for "units") often has to appear when you wouldn't expect it--in particular, when arithmetic is being done in a context that implies horizontal or vertical dimensions. For example,

    .ll 7i/2u

A safe rule is to attach a scale indicator to every number, even constants.

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

    .nr ll 7i/2
    .ll 0u

is sufficiently explicit as long as you use u with the .ll command.

Macros with Arguments

You can define macros that can change from one use to the next according to parameters supplied as arguments. To make this work, you need two things: first, when you define the macro, you must indicate that some parts of it will be provided as arguments when the macro is called. Second, when the macro is called you must provide actual arguments to be plugged into the definition.

To illustrate, let's define a macro .SM that will print its argument two points smaller than the surrounding text. The definition of .SM is

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

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

The following definition of .SM permits optional second and third arguments that will be printed in the normal size:

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

Arguments not provided when the macro is called are treated as empty. It is convenient to reverse the order of arguments because trailing punctuation is much more common than leading. The number of arguments that a macro was called with is available in number register $.

For example, let's define a macro .BD to create a bold Roman for **troff** command names in text. It combines horizontal motions, width computations, and argument rearrangement.

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

The \h and \w commands need no extra backslash, as we discussed earlier in this section. The \& is there in case the argument begins with a period.

Two backslashes are needed with the \\$n commands to protect one of them when the macro is being defined. Consider a macro called .SH which produces section headings rather like those in this paper, with the sections numbered automatically, and the title in bold in a smaller size. You would use it in this form:

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

If the argument to a macro is to contain spaces, then it must be surrounded by double quotation marks.

Here is the definition of the .SH macro:

```
.ta .75i 1.15i
.nr SH 0            \" initialize section number
.de SH
.sp 0.3i
.ft B
.nr SH \\n(SH+1     \" increment number
.ps \\n(PS-1        \" decrease PS
\\n(SH.  \\$1        \" number. title
.ps \\n(PS           \" restore PS
.sp 0.3i
.ft R
..
```

The section number is kept in number register SH, which is incremented each time just before it is used. Note that a number register may have the same name as a macro without conflict, but a string may not.

We used \\n(SH instead of \n(SH and \\n(PS instead of \n(PS. If we had used \n(SH, we would get the value of the register at the time the macro was defined, not at the time it was used. Similarly, by using \\n(PS, we get the point size at the time the macro is called.

As an example that does not involve numbers, recall the .NP macro which had a

        .tl 'left'center'right'

We could make these into parameters by using instead

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

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

        .ds CT - % -

but you can also supply private definitions for any of the strings.

Conditionals

To cause the .SH macro to leave two extra inches of space just before section 1, but nowhere else, you can put a test inside the .SH macro to determine whether the section number is 1, and add some space if it is. The .if command provides a conditional test just before the heading line is output:

        .if \\n(SH=1 .sp 2i   \" first section 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 if it were text. If the condition is false, or zero or negative, the rest of the line is skipped. It is possible to do more than one command if a condition is true. Suppose several operations are to be done before section 1. One possibility is to define a macro .S1 and invoke it if we are about to do section 1, as determined by an .if:.

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

An alternate way is to use the extended form of the .if, like this:

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

The braces \{ and \} must occur in the positions shown or you will get unexpected extra lines in your output.

Nroff and **troff** also provide an if-else construction. A condition can be negated by preceding it with !; we get the same effect as above by using

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

There are a handful of other conditions that can be tested with .if. For example, you may need to determine if the current page is even or odd. The following conditionals give facing pages different titles when used inside an appropriate new page macro.

```
.if e .tl "even page title"
.if o .tl "odd page title"
```

Two other conditions, which you will find useful when you need to process text for both lineprinter and typesetter, are n and t. These can be used to indicate conditions dependent on whether **troff** or **nroff** are being invoked.

```
.if t troff input ...
.if n nroff input ...
```

Finally, string comparisons may be made in an .if statement. The following comparison does "input" if string 1 is the same as string 2:

```
e&.if 'string1'string2' input
```

The character separating the strings can be anything reasonable that is not contained in either string. The strings themselves can reference strings with *, arguments with \$, and so on.

Environments

In an earlier section, the potential problem of going across a page boundary was mentioned: 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. **Nroff/troff** provides a way to deal with this and similar situations. There are three environments that have independently controllable versions of many of the parameters associated with processing, including size, font, line and title lengths, fill or no-fill mode, tab stops, and even partially collected lines. Thus the titling problem may be solved by processing the main text in one environment and titles in a separate environment with its own suitable parameters.

The environment command .ev *n* shifts to environment *n*; *n* must be 0, 1 or 2. The command .ev with no argument returns to the previous environment. Environment names are maintained in a stack, so calls for different environments may be nested and called in order. If, for example, the main text is processed in environment 0, which is where **troff** begins by default, we can modify the new page macro .NP to process titles in environment 1 like this:

```
.de NP
.ev 1                    \" shift to new environment
.lt 6i                   \" set parameters here
.ft R
.ps 10
... any other processing ...
.ev                      \" return to previous environment
..
```

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

Diversions

In page layout there are numerous occasions when it is necessary to store some text for a period of time without actually printing it. Footnotes are the most obvious example: the text of the footnote usually appears in the input long before the place on the page where it is to be printed is reached. In fact, the place where it is output normally depends on how big it is. The footnote text must be preprocessed at least to the extent that its size is determined.

Nroff and **troff** provide a mechanism called a diversion for doing this processing. Any part of the output may be diverted into a macro instead of being printed, and then at some convenient time the macro may be put back into the input. The command .di *xy* begins a diversion. All subsequent output is collected into the macro *xy* until the command .di with no arguments is encountered. This terminates the diversion. The processed text is available at any time thereafter, simply by giving the command:

```
.xy
```

The vertical size of the last finished diversion is contained in the built-in number register dn.

For example, suppose we want to implement a keep-release operation, so that text (such as a figure or table) between the commands .KS and .KE will not be split across a page boundary. Clearly, when a .KS is encountered, we have to begin diverting the output so we can find out how big it is. Then when a .KE is seen, we decide whether the diverted text will fit on the current page, and print it either there if it fits, or at the top of the next page if it doesn't. We could use the following to define .KS and .KE:

```
.de KS              \" start keep
.br                 \" start fresh line
.ev 1               \" collect in new environment
.fi                 \" make it filled text
.di XX              \" collect in XX
..
.de KE              \" end keep
.br                 \" get last partial line
.di                 \" end diversion
.if \\n(dn>=\\n(.t .bp   \" bp if doesn't fit
.nf                 \" bring it back in no-fill
.XX                 \" text
.ev                 \" return to normal environment
..
```

Recall that number register nl is the current position on the output page. Since output was being diverted, this remains at its value when the diversion started. The amount of text in the diversion is stored in dn. Another built-in register, .t is the distance to the next trap, which we assume 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 is issued automatically. In either case, the diverted output is then brought back with .XX. It is essential to bring it back in no-fill mode so **nroff/troff** will do no further processing on it.

The definition of .KS and .KE is only intended as an example to demonstrate the power of diversions. You will find the .KS and .KE macros already defined in the **MM** macro package.

Introduction

Nroff and **troff** are the XENIX text processing formatting programs. **Nroff** can be used to output text to terminals, lineprinters, and letter-quality printers. **Troff** can be used to output text to a number of phototypesetters and laser printers. Both programs use identical commands, which are interspersed with lines of text. The commands used by both programs allow you to control the style of headers and footers, footnotes, paragraphs, and sections. You may specify font and point size, spacing, multiple column output, and local motions to create overstriking and line drawing effects.

Because **nroff** and **troff** are compatible with each other, it is almost always possible to prepare input acceptable to both. By using conditional input, you may add commands which are specific to either program.

Invoking nroff and troff

The general form of invoking the formatters on the command line is:

> nroff *options files*

or

> troff *options files*

where *options* represents any of a number of option arguments and *files* represents a list of files containing the document to be formatted. An argument consisting of a single minus sign (-) is taken to be a filename corresponding to the standard input. If no filenames are given, input is taken from the standard input. The options may appear in any order so long as they appear before the filenames. They are:

-o*list*   Prints only pages whose page numbers appear in *list*, which consists of comma-separated numbers and number ranges. A number range has the form N-M and means pages N through M; an initial -N means from the beginning to page N, and a final N- means from N to the end.

-n*N*   Numbers first generated page N.

-s*N*   Stops every *N* pages. **Nroff** will halt prior to every *N* pages (default *N*=1) to allow paper loading or changing, and resume upon receipt of a newline. **Troff** will stop the phototypesetter every *N* pages, produce a trailer to allow changing cassettes, and will resume after the phototypesetter "start" button is pressed.

-m*name*   Prepends the macro file */usr/lib/tmac.name* to the input files.

-mc*name*   Same as -m*name*, but uses a compacted form of */usr/lib/tmac.name* for efficiency.

-ra*N*   Register a is set to *N*.

-i   Reads the standard input after the input files are exhausted.

-q   Invokes the simultaneous input-output mode of the rd request.

The following options are recognized by **nroff** only:

-T*name*   Specifies the name of the output terminal type.

-e   Produces equally-spaced words in adjusted lines, using full terminal resolution.

The following options are recognized by **troff** only:

-t   Directs output to the standard output instead of the phototypesetter.

-f   Refrains from feeding out paper and stopping phototypesetter at the end of the run.

-w   Waits until phototypesetter is available, if currently busy.

-b   Reports whether the phototypesetter is busy or available. No text processing is done.

-a   Sends a printable ASCII approximation of the results to the standard output.

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

Note that each option must be invoked as a separate argument.

Technical Information

The input to the formatters consists of text lines interspersed with control lines that set parameters or otherwise control later processing. Control lines begin with a "control character", usually a period (.) or a single quotation mark ('), followed by a one- or two-character name that specifies a basic "request" or the substitution of a user-defined "macro" in place of the control line. The single quotation mark control character (') suppresses the "break function," which is the forced output of a partially filled line caused by certain requests. The control character may be separated from the request or macro name by whitespace (spaces and/or tabs) for esthetic reasons. Names must be followed by either a space or a newline. Control lines with unrecognized names are ignored.

Various special functions may be introduced anywhere in the input by means of an "escape" character, normally the backslash (\). For example, the function "\nR" causes the interpolation of the contents of the number register R in place of the function; here R is either a single character name as in \nx, or a left-parenthesis-introduced, two-character name as in \n(xx.

Troff uses 432 units to the inch, corresponding to the Wang Laboratories phototypesetter which has a horizontal resolution of 1/432-inch and a vertical resolution of 1/144-inch. **Nroff** uses 240 units to the inch internally, corresponding to the least common multiple of the horizontal and vertical resolutions of various typewriter-like output devices. **Troff** rounds horizontal and vertical numerical parameter input to the actual horizontal and vertical resolution of the typesetter. **Nroff** similarly rounds numerical input to the actual resolution of the output device indicated by the -T option.

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, V is the current vertical line spacing in basic units, and C is a nominal character width in basic units, as shown below:

Scale Indicator	Meaning	Number of basic units Troff	Nroff
i	Inch	432	240
c	Centimeter	432x50/127	240x50/127
P	Pica = 1/6 inch	72	240/6
m	Em = S points	6xS	C
n	En = Em/2	3xS	C, same as Em
p	Point = 1/72 inch	6	240/72
u	Basic unit	1	1
v	Vertical line space	V	V
none	Default		

In **nroff**, both the em and the en are taken to be equal to the C, which is output-device dependent; common values are 1/10- and 1/12-inch. Actual character widths in **nroff** need not be all the same and constructed characters such as -> (→) are often extra wide. The default scaling is ems for the horizontally-oriented requests and functions, including:

    .ll .in .ti .ta .lt .po .mc \h \l;

Vs is the scaling for the vertically-oriented requests and the following functions:

    .pl .wh .ch .dt .sp .sv .ne .rt .ev \v \x \L

p is the scale for the .vs request; and u is the scale for the requests .nr, .if, and .ie. All other requests ignore any scale indicators. When a number register containing an

already appropriately scaled number is interpolated to provide numerical input, the unit scale indicator u may need to be appended to prevent an additional inappropriate default scaling. The number N may be specified in decimal-fraction form but the parameter finally stored is rounded to an integer number of basic units.

The "absolute" position indicator (|) may be prepended to a number N to generate the distance to the vertical or horizontal place N. For vertically oriented requests and functions, |N becomes the distance in basic units from the current vertical place on the page or in a "diversion" to the vertical place N. For all other requests and functions, |N becomes the distance from the current horizontal place on the input line to the horizontal place N.

For example,

.sp |3.2c

will space in the required direction to 3.2 centimeters from the top of the page. Wherever numerical input is expected, an expression involving parentheses, the arithmetic operators (+, -, /, *, %) and the logical operators (<, >, <=, >=, =, ==, & (and), : (or)) may be used. Except where controlled by parentheses, evaluation of expressions is left-to-right; there is no operator precedence. In the case of certain requests, an initial + or - is stripped and interpreted as an increment or decrement indicator respectively.

For example, if the number register x contains 2 and the current point size is 10, then

.ll (4.25i+0P+3)/2u

sets the line length to 1/2 the sum of 4.25 inches + 2 picas + 30 points.

Note: numerical parameters are indicated here in two ways. $\pm N$ means that the argument may take the forms N, +N, or -N and that the corresponding effect is to set the affected parameter to N, to increment it by N, or to decrement it by N respectively. Plain N means that an initial algebraic sign is not an increment indicator, but merely the sign of N. Generally, unreasonable numerical input is either ignored or truncated to a reasonable value. For example, most requests expect to set parameters to non-negative values; exceptions are .sp, .wh, .ch, .nr, and .if. The requests .ps, .ft, .po, .vs, .ls, .ll, .in and .lt restore the previous parameter value in the absence of an argument.

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

Basic Formatting Requests

The following sections describe the commonly used **nroff** and **troff** formatting requests.

Font and Character Size Control

The **troff** character set includes a regular character set plus a Special Mathematical Font character set--each having 102 characters. All ASCII characters are included, with some on the Special Font. With three exceptions, the ASCII characters are input as themselves, and non-ASCII characters are input in the form \(xx where xx is a two-character name. The three ASCII exceptions are mapped as follows:

ASCII Input Character	Printed by **troff** Name
´	acute accent
`	grave accent
–	minus

The characters ', ', and – may be input as \', \', and \- respectively or by their names. The ASCII characters @, #, ", ', ', <, >, \, {, }, ~, ^, and _ exist only on the Special Font and are printed as a 1-em space if that font is not mounted. **Nroff** understands the entire **troff** character set, but can in general print only ASCII characters, such characters as can be constructed by overstriking or other combinations, and those that can reasonably be mapped into other characters. The exact behavior is determined by a driving table prepared for each device. The characters ', ', and _ print as themselves. The default mounted fonts are Roman (R), italic (I), bold (B), and the Special Mathematical Font (S) on physical typesetter positions 1, 2, 3, and 4 respectively.

The current font, initially Roman, may be changed (among the mounted fonts) by use of the .ft request, or by imbedding at any desired point either \f$x$, \f(xx, or \f$N$ where x and xx are the name of a mounted font and N is a numerical font position. It is not necessary to change to the Special font; characters on that font are handled automatically. A request for a named but unmounted font is ignored. **Troff** can be informed that any particular font is mounted by use of the .fp request. The list of known fonts is installation-dependent. **Nroff** understands font control and normally underlines characters that are italicized.

Character point sizes are typically in the range 6-36 (1/12- to 1/2-inch). The .ps request is used to change or restore the point size. Alternatively the point size may be changed between any two characters by imbedding a \s$N$ at the desired point to set the size to N, or a \s±N($1 \le N \le 9$) to increment/decrement the size by N; \s0 restores the previous size. Requested point size values that are between two valid sizes yield the larger of the two. The current size is available in the .s register. **Nroff** ignores type size control.

A list of font and size control commands follows:

.ps   Has an initial value of 10. Point size set to ±N. Alternatively imbed \sN or \s±N. Any positive size value may be requested; if invalid, the next larger valid size will result, with a maximum of 36. A paired sequence +N, -N will work because the previous requested value is also remembered. Ignored in **nroff.** If no argument is given, .ps has the previous value.

.ss N   Has an initial value of 12/36 em. Space-character size is set to N/36 ems. This size is the minimum word spacing in adjusted text. Ignored in **nroff.** If no argument is specified, the request is ignored.

.cs F N M Initially off. Constant character space (width) mode is set on for font F (if mounted); the width of every character will be taken to be N/36 ems. If M is absent, the em is that of the character's point size; if M is given, the em is M points. All affected characters are centered in this space, including those with an actual width larger than this space. Special Font characters occurring while the current font is F are also so treated. If N is absent, the mode is turned off. The mode must be in effect when the characters are physically printed. Ignored in **nroff.**

.bd F N   Initially off. The characters in font F will be artificially emboldened by printing each one twice, separated by N-1 basic units. A reasonable value for N is 3 when the character size is in the vicinity of 10 points. If N is missing the embolden mode is turned off. The mode must be in effect when the characters are physically printed. Ignored in **nroff.**

.bd S F N Initially off. The characters in the Special Font will be emboldened whenever the current font is F. The mode must be in effect when the characters are physically printed.

.ft F   Initially Roman. Font changed to F. Alternatively, imbed \fF. The font name P is reserved to mean the previous font. If no argument is specified, previous font is assumed.

.fp N F   Initially R, I, B, S. Font position. This is a statement that a font named F is mounted on position N(1-4). It is a fatal error if F is not known. The phototypesetter has four fonts physically mounted. Each font consists of a film strip which can be mounted on a numbered quadrant of a wheel. This request is ignored if no arguments are given.

Page Control

Top and bottom margins are not automatically provided. It is standard procedure to define two macros and set traps for them at vertical positions 0 (top) and -N (N from

the bottom). A pseudo-page transition onto the first page occurs either when the first break occurs or when the first nondiverted text processing occurs. Arrangements for a trap to occur at the top of the first page must be completed before this transition.

.pl±N   Page length set to ±N, initially 11 inches. The internal limitation is about 75 inches in **troff** and about 136 inches in **nroff**. The current page length is available in the .p register. The default scale indicator is v. If no argument is given, 11 inches is assumed.

.bp±N   Begin page, initially N=1. The current page is ejected and a new page is begun. If ±N is given, the new page number will be ±N. The default scale indicator is v.

.pn±N   Page number, initially N=1. The next page (when it occurs) will have the page number ±N. A .pn must occur before the initial pseudo-page transition to effect the page number of the first page. The current page number is in the % register.

.po±N   Page offset, initially 0. The current left margin is set to ±N. The **troff** initial value provides about 1 inch of paper margin including the physical typesetter margin of 1/27-inch. In **troff** the maximum line-length + page-offset is about 7.54 inches. The current page offset is available in the .o register.

.ne N   Need N vertical space. If the distance D to the next trap position is less than N, a forward vertical space of size D occurs, which will spring the trap. If there are no remaining traps on the page, D is the distance to the bottom of the page. If D<V, another line could still be output and spring the trap. In a diversion, D is the distance to the diversion trap, if any, or is very large. If no argument is specified, N=1 V.

.mk R   Marks the current vertical place in an internal register (both associated with the current diversion level), or in register R, if given.

.rt±N   Returns upward only to a marked vertical place in the current diversion. If ±N is given, the place is ±N from the top of the page or diversion or, if N is absent, to a place marked by a previous .mk. Note that the .sp request may be used in all cases instead of .rt by spacing to the absolute place stored in an explicit register.

Text Filling, Adjusting, and Centering

Normally, words are collected from input text lines and assembled into an output text line until some word doesn't fit. An attempt is then made to hyphenate the word in an effort to place a part of it onto the output line. The spaces between the words on the output line are then increased to spread out the line to the current line length minus

any current indent. A word is any string of characters delimited by the space character or the beginning or end of the input line. Any adjacent pair of words that must be kept together (neither split across output lines nor spread apart in the adjustment process) can be tied together using the unpaddable space character (backslash-space). The adjusted word spacings are uniform in **troff** and the minimum interword spacing can be controlled with the .ss request. In **nroff**, word spacings are normally nonuniform because of quantization to character-size spaces; the command line option –e causes uniform spacing with full output device resolution. Filling, adjustment, and hyphenation can all be prevented or controlled. The text length on the last line output is available in the .n register, and text baseline position on the page for this line is in the .nl register. The text baseline high-water mark (lowest place) on the current page is in the .h register.

An input text line ending with ., ?, or ! is taken to be the end of a sentence, and an additional space character is automatically provided during filling. Multiple interword space characters found in the input are retained, except for trailing spaces; initial spaces also cause a break. When filling is in effect a \p may be embedded or attached to a word to cause a break at the end of the word and have the resulting output line spread out to fill the current line length.

A text input line that happens to begin with a control character can be printed as a text line by prefacing it with the nonprinting, zero-width filler character \&. Another method is to specify output translation of some convenient character into the control character using .tr.

The copying of an input line in no-fill mode can be interrupted by terminating the partial line with a \c. The next encountered input text line will be considered to be a continuation of the same line of input text. Similarly, a word within filled text may be interrupted by terminating the word and line with \c; the next encountered text will be taken as a continuation of the interrupted word. If the intervening control lines cause a break, any partial line will be forced out along with any partial word.

.br   Break. The filling of the line currently being collected is stopped and the line is output without adjustment. Text lines beginning with space characters and empty text lines (blank lines) also cause a break.

.fi   Fill subsequent output lines. Initially fill is on. The register .u is 1 in fill mode and 0 in nofill mode.

.nf   Nofill. Initially, fill is on. Subsequent output lines are neither filled nor adjusted. Input text lines are copied directly to output lines without regard for the current line length.

.ad c Line adjustment is begun. If fill mode is not on, adjustment will be deferred until fill mode is back on. If the type indicator c is present, the adjustment type is changed in the following ways: l to adjust left-margin only, r to adjust right margin only, c to center, b or n to adjust both margins. If c is absent the line remains unchanged.

.na   No-adjust. Initially, set to adjust. Adjustment is turned off; the right margin will be ragged. The adjustment type for .ad is unchanged. Output line filling still occurs if fill mode is on.

.ce *N* Initially off. Center the next *N* input text lines within the current line-length minus indent. If *N*=0, any residual count is cleared. A break occurs after each of the *N* input lines. If the input line is too long, it will be left-adjusted.

Vertical Spacing

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

If a word contains a vertically tall construct requiring the output line containing it to have extra vertical space before and or after it, the extra line space function \x'*N*' can be imbedded in or attached to that word. In this and other functions having a pair of delimiters around their parameter, the delimiter choice is arbitrary, except that it can't look like the continuation of a number expression for *N*. If *N* is negative, the output line containing the word will be preceded by *N* extra vertical space; if *N* is positive, the output line containing the word will be followed by *N* extra vertical space. If successive requests for extra space apply to the same line, the maximum values are used. The most recently utilized post-line extra line space is available in the .a register.

A block of vertical space is ordinarily requested using .sp, which honors the no-space mode and which does not space past a trap. A contiguous block of vertical space may be reserved using .sv. The following requests control vertical spacing:

.vs *N* Initially 1/6-inch or 12 points. Set vertical baseline spacing size V. Transient extra vertical space available with \x'*N*'.

.ls *N* Initially *N*=1. Line spacing set to ±*N*. Vs (blank lines) are appended to each output text line. Appended blank lines are omitted, if the text or previous appended blank line reached a trap position. Space vertically in either direction. If *N* is negative, the motion is backward (upward) and is limited to the distance to the top of the page. Forward (downward) motion is truncated to the distance to the nearest trap.

.sp N   Space vertically in either direction. If *N* is negative, the motion is backward (upward) and is limited to the distance to the top of the page. Forward (downward) motion is truncated to the distance to the nearest trap. If no-space mode is on, no spacing occurs.

.sv N   Save a contiguous vertical block of size *N*. If the distance to the next trap is greater than *N*, *N* vertical space is output. No-space mode has no effect. If this distance is less than *N*, no vertical space is immediately output, but *N* is remembered for later output. Subsequent .sv requests will overwrite any still remembered *N*.

.os   Output saved vertical space. No-space mode has no effect. Used to finally output a block of vertical space requested by an earlier .sv request.

.ns   No-space mode turned on. When on, the no-space mode inhibits .sp requests and .bp requests without a next page number. The no-space mode is turned off when a line of output occurs, or with .rs.

.rs   Restore spacing. The no-space mode is turned off.

blank line   Causes a break and output of a blank line exactly like .sp 1.

Line Length and Indenting

The maximum line length for fill mode may be set with .ll. The indent may be set with .in; an indent applicable to only the next output line may be set with the temporary indent request .ti. The line length includes indent space but not page offset space. The line length minus the indent is the basis for centering with .ce. The effect of .ll, .in, or .ti is delayed if a partially collected line exists, until after that line is output. In fill mode the length of text on an output line is less than or equal to the line length minus the indent. The current line length and indent are available in registers .l and .i respectively. The length of three-part titles produced by .tl is independently set by .lt.

.ll±N   Initially 6.5 inches. Line length is set to ±*N*. In **troff** the maximum line-length + page-offset is about 7.54 inches. Without an argument, this means the previous line length.

in±N   Initially *N*=0. Indent is set to ±*N*. The indent is prepended to each output line. Without an argument, this means the previous indent.

.ti±N   Temporary indent. The next output text line will be indented a distance ±*N* with respect to the current indent. The resulting total indent may not be negative. The current indent is not changed. Without an argument, the request is ignored.

Tabs, Leaders, and Fields

The ASCII horizontal tab character and the ASCII SOH character (leader) can both be used to generate either horizontal motion or a string of repeated characters. The length of the generated entity is governed by internal tab stops specifiable with .ta. The default difference is that tabs generate motion and leaders generate a string of periods; .tc and .lc offer the choice of repeated character or motion. There are three types of internal tab stops: left adjusting, right adjusting, and centering. In the following table D is the distance from the current position on the input line (where a tab or leader was found) to the next tab stop; the next string consists of the input characters following the tab (or leader) up to the next tab (or leader) or end of line; and W is the width of next-string.

Tab type	Length of motion or repeated characters	Location of next string
Left	D	Following D
Right	D-W	Right adjusted within D
Centered	D-W/2	Centered on right end of D

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

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

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 incorporated as horizontal padding space that is divided among the indicated padding places. The incorporated padding is allowed to be negative. For example, if the field delimiter is # and the padding indicator is ^, #^xxx^right# specifies a right-adjusted string with the string *xxx* centered in the remaining space. The following requests are recognized:

.ta Nt ...   Sets tab stops and types. N=number; t=R, right adjusting; t=C, centering; t absent is left-adjusting. **Troff** tab stops are preset every 0.5 inches, **nroff** every 0.8 inches. The stop values are separated by spaces, and a value preceded by + is treated as an increment to the previous stop value.

.tc c   The tab repetition character becomes c, or is removed specifying motion.

.lc c   The leader repetition character becomes c, or is removed specifying motion.

.fc a b   The field delimiter is set to a; the padding indicator is set to the space
            character or to b, if given. In the absence of arguments the field mechanism
            is turned off.

Hyphenation

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

Only words that consist of a central alphabetic string surrounded by (usually null)
nonalphabetic strings are considered candidates for automatic hyphenation. Words that
were input containing hyphens (minus), em-dashes (\(em), or hyphenation indicator
characters--such as mother-in-law--are always subject to splitting after those
characters, whether automatic hyphenation is on or off.

.nh   Initially hyphenation is on. Automatic hyphenation is turned off.

.hy N   Automatic hyphenation is turned on for $N \geq 1$, or off for $N=0$. If $N=2$, last
            lines (ones that will cause a trap) are not hyphenated. For $N=4$ and 8, the
            last and first two characters respectively of a word are not split off. These
            values are additive; i.e., $N=14$ will invoke all three restrictions.

.hc c   Hyphenation indicator character is set to c or to the default \&. The
            indicator does not appear in the output.

.hw word1...
            Specify hyphenation points in words with imbedded minus signs. Versions of
            a word with various endings are implied.

Three Part Titles

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

.tl'left'center'right'
            The strings left, center, and right are respectively left-adjusted, centered,
            and right-adjusted in the current title length. Any of the strings may be
            empty, and overlapping is permitted. If the page-number character (initially

%) is found within any of the fields it is replaced by the current page number having the format assigned to the register %. Any character may be used as the string delimiter.

.pc c   The page number character is set to c, or removed. The page-number register remains %.

.lt ±N   Initially 6.5 inches. Length of title set to ±N. The line length and the title length are independent. Indents do not apply to titles; page offsets do.

Output Line Numbering

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

.nm ±N   Line number mode. If ±N is given, line numbering is turned on, and the next output line numbered is numbered ±N. Default values are M=1, S=R, and I=0. Parameters corresponding to missing arguments are unaffected; a non-numeric argument is considered missing. In the absence of all arguments, numbering is turned off; the next line number is preserved for possible further use in number register ln.

.nn N   The next N text output lines are not numbered.

Character Translations, Overstrike, and Local Motions

The **troff** functions described in the following sections apply to the processing of specialized text, including special characters and lines of variable length. Also described are methods for producing special effects in text, by changing the position of text relative to lines and using offsets to create bold effects.

Input/Output Conventions and Character Translations

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

The **troff** escape character backslash (\) introduces escape sequences--causes the following character to mean another character, or to indicate some function. The backslash (\) should not be confused with the ASCII control character ESC of the same name. The escape character \ can be input with the sequence \\. The escape character can be changed with .ec, and all that has been said about the default \ becomes true for the new escape character. The sequence \e can be used to print whatever the current escape character is. If necessary or convenient, the escape mechanism may be turned off with .eo, and restored with .ec.

.ec c   Sets escape character to \, or to c, if given.

.eo   Turns the escape mechanism off.

Five ligatures are available in the current **troff** character set: fi, fl, ff, Fi, and ffl. They may be input in **nroff** with \(fi, \(fl, \(ff, \(Fi, and \(Fl respectively.

The ligature mode is normally on in **troff**, and automatically invokes ligatures during input. The ligature request is:

.lg N   Ligature mode is turned on if N is absent or nonzero, and turned off if $N=0$. If $N=2$, only the two-character ligatures are automatically invoked. Ligature mode is inhibited for request, macro, string, register, or filenames, and in copy mode. No effect in **nroff**.

Unless in copy mode, the ASCII backspace character is replaced by a backward horizontal motion having the width of the space character. **Nroff** automatically underlines characters in the underline font, specifiable with uf, normally on font position 2. In addition to .ft and \fF, the underline font may be selected by .ul and .cu. Underlining is restricted to an output-device-dependent subset of reasonable characters.

.ul N   Initially off. Underlines in **nroff** (italicizes in **troff**) the next N input text lines. Actually, switches to underline font, saving the current font for later restoration; other font changes within the span of a .ul will take effect, but the restoration will undo the last change. Output generated by .tl is affected by the font change, but does not decrement N. If $N>1$, there is the risk that a trap interpolated macro may provide text lines within the span; environment switching can prevent this.

.cu N   Initially off. A variant of .ul that causes every character to be underlined in **nroff**. Identical to .ul in **troff**.

.uf F   Initially italic. Underline font set to F. In **nroff**, F may not be on position 1.

Both the control character dot (.) and the no-break control character (') may be changed, if desired. Such a change must be compatible with the design of any macros used in the span of the change, and particularly of any trap-invoked macros.

.cc c   The basic control character is set to c, or reset to dot (.).

.c2 c   The nobreak control character is set to c, or reset to single quotation mark (').

One character can be made to stand in for another character using .tr. All text processing (e.g., character comparisons) takes place with the input (stand-in) character, which appears to have the width of the final character. The graphic translation occurs at the moment of output (including diversion).

.tr abcd..   Translates a to b, c to d, etc. If an odd number of characters is given, the last one will be mapped into the space character. To be consistent, a particular translation must stay in effect from input to output time.

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

Comments and concealed newlines may appear in text. An uncomfortably long input line that must stay one line (e.g., a string definition, or nofilled text) can be split into many physical lines by ending all but the last one with the escape \. The sequence \(newline) is always ignored—except in a comment. Comments may be imbedded at the end of any line by prefacing them with \". The newline at the end of a comment cannot be concealed. A line beginning with \" will appear as a blank line and behave like .sp 1; a comment can be on a line by itself if the line begins with .\".

Local Motions and the Width Function

The functions \v'N' and \h'N' can be used for local vertical and horizontal motion respectively. The distance N may be negative; the positive directions are rightward and downward. A local motion is one contained within a line, and otherwise within a line balance to zero. The vertical motions are:

\v'N'   Move distance N

\u   1/2-em up in **troff**; 1/2-line up in **nroff**

\d   1/2-em down in **troff**; 1/2-line down in **nroff**

\r   1 em up in **troff**; 1 line up in **nroff**

The horizontal motions are:

\h'*N*'	Move distance *N*
space	Unpaddable space-size space
\0	Digit-sized space
\\|	1/6-em space in **troff**; ignored in **nroff**
\^	1/12-em space in **troff**; ignored in **nroff**

The width function \w'string' generates the numerical width of *string* (in basic units). Size and font changes may be safely imbedded in *string*, and will not affect the current environment. For example, .ti-w'1.'u could be used to temporarily indent leftward a distance equal to the size of the string "1."

The width function also sets three number registers. The registers st and sb are set to the highest and lowest extent of string relative to the baseline; then, for example, the total height of *string* is \n(stu-\n(sbu. In **troff** the number register ct is set to a value between 0 and 3: 0 means that all of the characters in string were short lowercase characters without descenders (e.g., c); 1 means that at least one character has a descender (e.g., y); 2 means that at least one character is tall (e.g., H); and 3 means that both tall characters and characters with descenders are present. The escape sequence \kx will cause the current horizontal position in the input line to be stored in register x.

Overstrike, Bracket, Line-drawing, and Zero-width Functions

Automatically centered overstriking of up to nine characters is provided by the overstrike function \o'*string*'. The characters in *string* are overprinted with centers aligned; the total width is that of the widest character. *String* should not contain local vertical motion. The function \zc will output c without spacing over it, and can be used to produce left-aligned overstruck combinations.

The Special Mathematical Font contains a number of bracket construction pieces

that can be combined into various bracket styles. The function \b'*string*' may be used to pile the characters in *string* vertically (the first character on top and the last at the bottom); the characters are vertically separated by 1 em and the total pile is centered 1/2-em above the current baseline.

The function \l'Nc' will draw a string of repeated c's towards the right for a distance N. (\l is \(lowercase L). If c looks like a continuation of an expression for N, it may be insulated from N with a \&. If c is not specified, the _ (baseline rule) is used (underline character in **nroff**). If N is negative, a backward horizontal motion of size N is made before drawing the string. Any space resulting from N/(size of c) having a remainder is

put at the beginning (left end) of the string. In the case of characters that are designed to be connected such as baseline-rule (_), underrule (_), and root-en (⁻), the remaining space is covered by overlapping. If N is less than the width of c, a single c is centered on a distance N.

The function \L'Nc' will draw a vertical line consisting of the (optional) character c stacked vertically apart 1 em (1 line in **nroff**) with the first two characters overlapped, if necessary, to form a continuous line. The default character is the box rule (\(br); the other suitable character is the bold vertical (\(bv). The line is begun without any initial motion relative to the current base line. A positive N specifies a line drawn downward and a negative N specifies a line drawn upward. After the line is drawn no compensating motions are made; the instantaneous baseline is at the end of the line. The horizontal and vertical line drawing functions may be used in combination to produce large boxes. The zero-width box-rule and the 1/2-em wide underrule were designed to form corners when using 1 em vertical spacings.

Processing Control Facilities

The following sections describe **nroff** and **troff** requests and facilities for controlling the processing of text.

Macros, Strings, Diversions, and Position Traps

A "macro" definition is a named set of arbitrary lines that may be invoked by name or with a trap. A "string" is a named string of characters, not including a newline character, that may be interpolated by name at any point. Request, macro, and string names share the same name list. Macro and string names may be one or two characters long and may usurp previously defined request, macro, or string names. Any of these may be renamed with .rn or removed with .rm. Macros are created by .de and .di, and appended to by .am and .da; .di and .da cause normal output to be stored in a macro. Strings are created by .ds and appended to by .as. A macro is invoked in the same way as a request; a control line beginning with .xx will interpolate the contents of macro xx. The remainder of the line may contain up to nine arguments. The strings x and xx are interpolated at any desired point with *x and *(xx respectively. String references and macro invocations may be nested.

During the definition and extension of strings and macros (not by diversion) the input is read in copy mode. The input is copied without interpretation except that:

- The contents of number registers indicated by \n are interpolated.

- Strings indicated by * are interpolated.

- Arguments indicated by \$ are interpolated.

- Concealed newlines indicated by \newline are eliminated.

- Comments indicated by \" are eliminated.

- \t and \a are interpreted as ASCII horizontal tab and SOH respectively.

- \\ is interpreted as \.

- \. is interpreted as dot (.).

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

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

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

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

and called with

```
.xx Monday 14th
```

to produce the text

    Today is Monday the 14th.

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

No arguments are available at the top (nonmacro) level in this implementation. Because string referencing is implemented as an input-level push down, no arguments are available from within a string. No arguments are available within a trap-invoked macro.

Arguments are copied in copy mode onto a stack where they are available for reference. The mechanism does not allow an argument to contain a direct reference to a long string (interpolated at copy time) and it is advisable to conceal string references (with an extra \) to delay interpolation until argument reference time.

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

Diversions may be nested and certain parameters and registers are associated with the current diversion level (the top nondiversion level may be thought of as the 0th diversion level). These are the diversion trap and associated macro, the no-space mode, the internally saved marked place (see .mk and .rt), the current vertical place (.d register), the current high-water text baseline (.h register), and the current diversion name (.z register).

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

A macro-invocation trap effective in the current diversion may be planted using .dt. The .t register works in a diversion; if there is no subsequent trap a large distance is returned. For a description of input line count traps, see .it in the following list.

.de xx yy   Define or redefine the macro xx. The contents of the macro begin on the next input line. Input lines are copied in copy mode until the definition is terminated by a line beginning with .yy, whereupon the macro yy is called. In the absence of yy, the definition is terminated

by a line beginning with two dots (..). A macro may contain .de requests provided the terminating macros differ or the contained definition terminator is concealed. The dots can be concealed as \\.. which will copy as \.. and be reread as dots (..).

.am *xx yy*   Append to macro.

.ds *xx string*   Define a string *xx* containing string. Any initial double quotation mark in string is stripped off to permit initial blanks.

.as *xx string*   Append string to string *xx*.

.rm *xx*   Remove request, macro, or string. The name *xx* is removed from the name list and any related storage space is freed.   Subsequent references will have no effect.

.rn *xx yy*   Rename request, macro, or string *xx* to *yy*. If *yy* exists, it is first removed.

.di *xx*   Divert output to macro *xx*.   Normal text processing occurs during diversion except that page offsetting is not done. The diversion ends when the request .di or .da is encountered without an argument; extraneous requests of this type should not appear when nested diversions are being used.

.da *xx*   Divert, appending to *xx*.

.wh *N xx*   Install a trap to invoke *xx* at page position *N*; a negative *N* will be interpreted with respect to the page bottom. Any macro previously planted at *N* is replaced by *xx*. A zero *N* refers to the top of a page. In the absence of *xx*, the first found trap at *N*, if any, is removed.

.ch *xx N*   Change the trap position for macro *xx* to *N*. In the absence of *N*, the trap is removed.

.dt *N xx*   Install a diversion trap at position *N* in the current diversion to invoke macro *xx*.   Another .dt will redefine the diversion trap.   If no arguments are given, the diversion trap is removed.

.it *N xx*   Set an input line count trap to invoke the macro *xx* after *N* lines of text input have been read (control or request lines don't count). The text may be in-line text or text interpolated by in-line or trap-invoked macros.

.em *xx*   The macro *xx* will be invoked when all input has ended. The effect is the same as if the contents of *xx* had been at the end of the last file processed.

Number Registers

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

Number registers are created and modified using .nr, which specifies the name, numerical value, and the auto-increment size. Registers are also modified, if accessed with an auto-incrementing sequence. If the registers x and xx both contain N and have the auto-increment size M, the following access sequences have the effect shown:

Sequence	Effect on Register	Value Interpolated
\nx	none	N
\n(xx	none	N
\n+x	x incremented by M	N+M
\n-x	x decremented by M	N-M
\n+(xx	xx incremented by M	N+M
\n-(xx	xx decremented by M	N-M

When interpolated, a number register is converted to decimal (default), decimal with leading zeros, lowercase Roman, uppercase Roman, lowercase sequential alphabetic, or uppercase sequential alphabetic according to the format specified by .af.

.nr R±NM   The number register R is assigned the value ±N with respect to the previous value, if any. The increment for auto-incrementing is set to M.

.af R c   Assign format c to register R. The available formats are:

Format	Numbering Sequence
1	0,1,2,3,4,5,...
001	000,001,002,003,004,005,...
i	0,i,ii,iii,iv,v,...
I	0,I,II,III,IV,V,...
a	0,a,b,c,...,z,aa,ab,...,zz,aaa,...
A	0,A,B,C,...,Z,AA,AB,...,ZZ,AAA,...

An Arabic format having *N* digits specifies a field width of *N* digits.   The read-only registers and the width function are always Arabic.

.rr R   Remove register R. If many registers are being created dynamically, it may become necessary to remove no longer used registers to recapture internal storage space for newer registers.

Conditional Acceptance of Input

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

.if c *text*
          If condition c is true, process *text* as input; in multiline case, use \{text\}.

.if !c *text*
          If condition c is false, process *text*.

.if N *text*
          If expression N>0, process *text*.

.if !N *text*
          If expression N<0, process *text*.

.if 'string1'string2' *text*
          If string1 identical to string2, process *text*.

.if!'string1'string2' *text*
          If string 1 not identical to string2, process *text*.

.ie c *text*
          "If" portion of if-else; all above forms (like if).

.el *text*
          "Else" portion of if-else.

There are several built-in condition names:

o   Current page number is odd

e   Current page number is even

t   Formatter is **troff**

n   Formatter is **nroff**

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

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

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

Environment Switching

A number of the parameters that control text processing are gathered together into an environment, which can be switched by the user. Partially collected lines and words are in the environment. Everything else is global; examples are page-oriented parameters, diversion-oriented parameters, number registers, and macro and string definitions. All environments are initialized with default parameter values.

.ev N   Initially N=0. Environment switched to environment where N is in the range 0-2. Switching is done in push-down fashion so that restoring a previous environment must be done with .ev with no parameters rather than a specific numeric reference.

Insertions from the Standard Input

The input can be temporarily switched to the system standard input with .rd, which will switch back when two newlines in a row are found (the extra blank line is not used). This mechanism is intended for insertions in documentation containing standard formats. The standard input can be the terminal, a pipe, or a file.

.rd prompt   Reads insertion from the standard input until two newlines in a row are found. If the standard input is the user's keyboard, a prompt (or a BEL) is written onto the terminal. The .rd request behaves like a macro, and arguments may be placed after the prompt.

.ex   Exit from either **nroff** or **troff**. Text processing is terminated exactly as if all input had ended.

If insertions are to be taken from the terminal keyboard while output is being printed on the terminal, the command line option -q will turn off the echoing of keyboard input and prompt only with BEL. The regular input and insertion input cannot simultaneously come from the standard input.

Input/Output File Switching

The following requests control the switching of input and output files:

.so *filename*   Switch source file. The top input (file reading) level is switched to *filename*. The effect of a .so encountered in a macro is not felt until the input level returns to the file level. When the new file ends, input is again taken from the original file. .so's may be nested.

.nx *filename*   Next file is *filename*. The current file is considered ended, and the input is immediately switched to *filename*.

.pi *program*   Pipe output to *program* in **nroff** only. This request must occur before any printing occurs. No arguments are transmitted to *program*.

Miscellaneous Requests

.mc *c N*   Specifies that a margin character *c* appear a distance *N* to the right of the right margin after each nonempty text line (except those produced by .tl). If the output line is too long, the character will be appended to the line. If *N* is not given, the previous *N* is used; the initial *N* is 0.2 inches in **nroff** and 1 em in **troff**.

.tm *string*   After skipping initial blanks, *string* (rest of line) is read in copy mode and written on the user's terminal.

.ig *yy*   Ignores input lines. The .ig request behaves exactly like .de except that the input is discarded. The input is read in copy mode, and any auto-incremented registers will be affected.

.pm *t*   Prints macros. The names and sizes of all of the defined macros and strings are printed on the user's terminal; if *t* is given, only the total of the sizes is printed. The sizes are given in blocks of 128 characters.

.fl   Flushes output buffer. Used in interactive debugging to force output.

Output and Error Messages

The output from .tm, .pm, and the prompt from .rd, as well as various error messages are written onto the standard message output. The latter 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.

Various error conditions may occur during the operation of **nroff** and **troff.** Certain less serious errors having only local impact do not cause processing to terminate. Two examples are word overflow, caused by a word that is too large to fit into the word buffer (in fill mode), 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 offending excess is discarded, and the affected word or line is marked at the point of truncation with a * in **nroff** and a ⇐ in **troff.** The program continues processing, if possible, on the grounds that output useful for debugging may be produced. If a serious error occurs, processing terminates, and an appropriate message is printed. Examples are the inability to create, read, or write files, and the exceeding of certain internal limits that make future output unlikely to be useful.

Summary of Escape Sequences and Number Registers

Escape Sequences for Characters, Indicators, and Functions

Sequence	Meaning
\\	\(to prevent or delay the interpretation of \)
\e	Printable version of the current escape character
\'	Acute accent ('); equivalent to \(aa
\`	Grave accent (`); equivalent to \(ga
\-	Minus sign (-) in the current font
\.	Period (.)
\(space	Unpaddable space-size space character
\0	Digit width space
|	1/6-em narrow space character (zero width in **nroff**)
\^	1/12-em half-narrow space character (zero width in **nroff**)
\&	Nonprinting, zero-width character
\!	Transparent line indicator
\"	Beginning of comment
\$N	Interpolate argument $(1 \le N \le 9)$
\%	Default optional hyphenation character
\(xx	Character named xx
\a	Noninterpreted leader character
\b'abc...'	Bracket building function
\c	Interrupt text processing
\d	Forward (down) 1/2 em vertical motion (1/2 line in **nroff**)
\fx, \f(xx, \fN	Change to font named x or xx, or position N
\h'N'	Local horizontal motion; move right N (negative left)
\kx	Mark horizontal input place in register x
\l'Nc'	Horizontal line drawing function (optionally with c)
\L'Nc'	Vertical line drawing function (optionally with c)
\nx, \n(xx	Interpolate number register x or xx
\o'abc...'	Overstrike characters a, b, c
\p	Break and spread output line
\r	Reverse 1 em vertical motion (reverse line in **nroff**)
\sN, \s±N	Point size change function
\t	Noninterpreted horizontal tab
\u	Reverse 1/2-em vertical motion (1/2-line in **nroff**)
\v'N'	Local vertical motion; move down N (negative up)
\w'string'	Interpolate width of string
\x'N'	Extra line space function (negative before, positive after)
\zc	Print c with zero width (without spacing)
\{	Begin conditional input
\}	End conditional input
\(newline)	Concealed (ignored) newline
\X	X, any character not listed above

Predefined General Number Registers

Register	Contents
%	Current page number
ct	Character type (set by width function)
dl	Width (maximum) of last completed diversion
dn	Height (vertical size) of last completed diversion
dw	Current day of the week (1-7)
dy	Current day of the month (1-31)
hp	Current horizontal place on input line
ln	Output line number
mo	Current month (1-12)
nl	Vertical position of last printed text baseline
sb	Depth of string below base line (generated by width function)
st	Height of string above base line (generated by width function)
yr	Last two digits of current year

Predefined Read-Only Number Registers

Register	Contents
.&	Number of arguments available at the current macro level
.A	Set to 1 in **troff**, if -a option used; 1 in **nroff**
.H	Available in horizontal resolution in basic units
.T	Set to 1 in **nroff**, if -T option used; always 0 in **troff**
.V	Available vertical resolution in basic units
.a	Post-line extra line space most recently utilized using \x'N'
.c	Number of lines read from current input file
.d	Current vertical place in current diversion; equal to nl, if no diversion
.f	Current font as physical quadrant
.h	Text baseline high-water mark on current page or diversion
.i	Current indent
.l	Current line length
.n	Length of text portion on previous output line
.o	Current page offset
.p	Current page length
.s	Current point size
.t	Distance to the next trap
.u	Equal to 1 in fill mode and 0 in no-fill mode
.v	Current vertical line spacing
.w	Width of previous character
.x	Reserved version-dependent register
.y	Reserved version-dependent register
.z	Name of current diversion

Introduction

By now, you have a firm grasp of most of the principles and techniques of using XENIX text processing successfully. By using the **mm** macro package, along with **nroff/troff** commands, you should be able to achieve precise control of almost any formatting task. However, there are two formatting needs which may be best met with two specialized XENIX formatting programs:

- Formatting tables or other complicated multicolumn material

- Setting mathematical equations

In this chapter, the program **tbl**, the table formatting program, is introduced. **Eqn**, the mathematics formatting program, is discussed in Chapter 8. Unless you anticipate using tables or equations fairly extensively in your work, you may wish to postpone or skip reading about **tbl** and **eqn**. Although both programs use commands which are easy to learn and use, you should expect to spend several hours on each program--reading these instructions, learning the commands, and testing them out with your output device. If you need to create tables or equations in your documents, the effort of learning **tbl** and **eqn** will be well rewarded. You will be able to produce high-quality, consistent output with relatively little work.

Both **tbl** and **eqn** are "preprocessors"--that is, you insert commands into your text as you are preparing it, just as you would if you were using **mm**. These commands are translated by the **tbl** and **eqn** programs into sequences of **nroff/troff** commands, without altering either the body of your text or other formatting commands. Your file is then processed through the **nroff** or **troff** programs themselves.

You will find **tbl** especially useful in preparing charts, multicolumn list summaries, and other tabular material. It will give you a high degree of control over complicated column alignment, and it will calculate the necessary widths of columns, when the elements are of varying lengths. **Tbl** also allows you to draw horizontal lines, vertical lines and boxes in order to highlight your material. Although the effects will be somewhat limited if you are working with an ordinary lineprinter or similar device, you will obtain extremely high quality results when outputting tables to phototypesetter.

Because the **tbl** program works by isolating the tabular material from the rest of the file, and then creating the necessary **nroff** or **troff** commands, the rest of the file is left intact for other programs to format. Thus you can use **tbl** along with the equation formatting program **eqn** or various layout macro packages like **mm**, without duplicating their functions. You need only be careful to invoke the various programs in the correct order.

The latter part of this chapter is devoted to some examples--in each case, the text input is paired with the resulting output. You may find that at first you learn the features of **tbl** best by examining these examples and copying those formatting instructions for examples which resemble your own tables. However, first read the rules for preparing **tbl** input, so you have a general idea of how to invoke the **tbl** program, and an overview of the possible options and formats.

Input Format

The input to **tbl** is text for a document, with tables preceded by a .TS (table start) command and followed by a .TE (table end) command. **Tbl** processes the text and formatting commands within these two commands, generating **nroff/troff** formatting commands. The .TS and .TE lines are also copied so that **nroff** and **troff** page layout macros can use these lines to delimit and place tables as necessary. In particular, any arguments on the .TS or .TE lines are copied but otherwise ignored, and may be used by document layout macro commands.

The format of the input is:

        text
        .TS
        table
        .TE
        text
        .TS
        table
        .TE
        text

Each table will contain text, options, and formatting specifications:

        .TS
        options;
        format.
        data
        .TE

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 options that affect the entire table.

Each table may contain global options, a format section describing the layout of individual table entries, and then the text to be printed. The format and data are always required, but not the options. The various parts of a table are described in the following sections.

Options

There may be a single line of options which affect the whole table. If present, this line must immediately follow the .TS line and must contain a list of option names separated by spaces, tabs, or commas, and must be terminated by a semicolon. The allowable options 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 x instead of tab to separate data items

linesize (*n*)
>   Sets lines or rules in *n* point type

delim (*xy*)
>   Recognizes *x* and *y* as the eqn delimiters

The **tbl** program tries to keep boxed tables on one page by issuing appropriate *.ne* commands. These requests are calculated from the number of lines in the tables, and if there are spacing commands embedded in the input, these requests may be inaccurate. To ensure the correct format on one page, you can surround the table with the display macros .DS and .DE.

Format

The format section of the table specifies the layout of the columns. Each line in this section corresponds to one line of the table. The last format line applies to all the remaining lines in the table. Each line contains a keyletter for each column of the table. It is good practice to separate the key letters for each column by spaces or tabs. The keyletters, which may be either uppercase or lowercase, are:

L or l
>   Indicates a left-adjusted column entry

R or r
>   Indicates a right-adjusted column entry

C or c
>   Indicates a centered column entry

N or n
>   Indicates a numerical column entry, to be aligned with other numerical entries so that the units digits of numbers line up

A or a
>   Indicates an alphabetic column; all corresponding entries are aligned on the left, and positioned so that the widest is centered within the column

S or s
>   Indicates a spanned heading, i.e., the entry from the previous column continues across this column

^
>   Indicates a vertically spanned heading, i.e., the entry from the previous row continues down through this row. (Not allowed for the first row of the table.)

When you are aligning numerical information, a location for the decimal point is sought. The rightmost dot adjacent to a digit is used as a decimal point; if there is no dot adjoining a digit, the rightmost digit is used for the units; if no alignment is indicated, the item is centered in the column. However, the special nonprinting character string "\&" may be used to override dots and digits, or to align alphabetic data; this string lines up where a dot normally would, then disappears from the final output. In the example below, the items shown at the left will be aligned (in a numerical column) as shown on the right:

input	tbl format
13	13
4.2	4.2
26.12	26.12
abc	abc
abc\&	abc
43\&3.22	433.22
749.12	749.12

Note that if numerical text is used in the same column with wider left-adjusted (L) or right-adjusted (R) type table entries, the widest number is centered relative to the wider left-adjusted or right-adjusted items (L is used instead of 1 for readability; they have the same meaning as keyletters). Alignment within the numerical items is preserved, in the same way as using the A format. However, alphabetic subcolumns requested by the keyletter are always slightly indented relative to left-adjusted L items; if necessary, the column width is increased to force this. This is not true for n type entries. Do not put N and A type entries in the same column.

To make your table formatting information more readable, you should separate the keyletters describing each column with spaces. The layout of the keyletters in the format section resembles the layout of the actual data in the table. The end of the format section of the table specification is indicated by a period. For example, a simple format might look like this:

```
c s s
l n n.
```

This specifies a table of three columns. 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.

Here is a sample table in this format:

	Overall title	
Item-a	34.22	9.1
Item-b	12.65	.02
Items: c,d,e	23	5.8
Total	69.87	14.92

Note that instead of listing the format of successive lines of a table on consecutive lines of the format section, successive line formats may be given on the same line, separated by commas. In the example above, the format might have been written:

```
c s s, l n n.
```

Additional Features

There are some additional features of the keyletter system:

Horizontal Lines

A keyletter may be replaced by an underscore (_) to indicate a horizontal line in place of the corresponding column entry, or by an equal sign (=) to indicate a double horizontal line. If an adjacent column contains a horizontal line, or if there are vertical lines adjoining this column, this horizontal line is extended to meet the nearby lines. If any data entry is provided for this column, it is ignored and a warning message is printed.

Vertical Lines

A vertical bar (|) may be placed between column keyletters. This will cause a

vertical line between the corresponding columns of the table. 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.

Space Between Columns

A number may follow the keyletter. This indicates the amount of separation between this column and the next column. The number normally specifies the separation in ens (one en is about the width of the letter n), or more precisely, an en is a number of points (1 point = 1/72-inch) equal to half the current type size. If the "expand" option is used, then these numbers are multiplied by a constant so that the table is as wide as the current line length. The default column separation number is 3. If the separation is changed, the largest space requested prevails.

Vertical Spanning

Normally, vertically spanned 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.

Font Changes

A keyletter may be followed by a string containing a font name or number preceded by the letter f or F. This indicates that the corresponding column should be in a different font from the default font (usually Roman). All font names are one or two letters; a one-letter font name should be separated from whatever follows by a space or tab. Font change commands given with the table entries will override these specifications.

Point Size Changes

A keyletter may be followed by the letter p or P and a number to indicate the point size of the corresponding table entries. The number may be a signed digit, in which case it is taken as an increment or decrement from the current point size. If both a point size and a column separation value are given, one or more blanks must separate them.

Vertical Spacing Changes

A keyletter may be followed by the letter v or V and a number to indicate the vertical line spacing to be used within a multiline corresponding table entry. The number may be a signed digit, in which case it is taken as an increment or decrement from the current vertical spacing. A column separation value must be

separated by blanks or some other specification from a vertical spacing request. This request has no effect unless the corresponding table entry is a text block.

Column Width Indication

A keyletter may be followed by the letter w or W and a width value in parentheses. This width is used as a minimum column width. If the largest element in the column is not as wide as the width value given, the largest element is assumed to be that wide. If the largest element in the column is wider than the specified value, its width is used. The width is also used as a default line length for included text blocks. Normal **troff** units can be used to scale the width value; the default is ens. If the width specification is a unitless integer the parentheses may be omitted. If the width value is changed in a column, the last value given controls.

Equal Width Columns

A keyletter may be followed by the letter e or E to indicate equal width columns. All columns whose keyletters are followed by e or E are made the same width. This allows you to get a group of regularly spaced columns.

The order of the above features is immaterial; they need not be separated by spaces, except as indicated above to avoid point size and font change ambiguities. Thus a numerical column entry in italic font and 12-point type with a minimum width of 2.5 inches and separated by 6 ens from the next column could be specified as

np12w(2.5i)fI 6

Note the following format defaults: Column descriptors missing from the end of a format line are assumed to be L. The longest line in the format section, however, defines the number of columns in the table; extra columns in the data are ignored silently.

Data

The text for the table is typed after the format specification. Normally, each table line is typed as one line of data. Very long input lines can be broken: any line whose last character is a backslash (\) is combined with the following line (and the backslash vanishes). The data for different columns (the table entries) are separated by tabs, or by whatever character has been specified in the tabs option. There are a few special cases:

Troff commands within tables

An input line beginning with a dot (.) followed by anything but a number is assumed

to be a command to **troff** and is passed through unchanged, retaining its position in the table. For example space within a table may be produced by .sp commands in the data.

Full Width Horizontal Lines

An input line containing only the underscore (_) or equal sign (=) is taken to be a single or double line, respectively, extending the full width of the table.

Single Column Horizontal Lines

An input table entry containing only the underscore or equal sign character is taken to be a single or double line extending the full width of the column. Such lines are extended to meet horizontal or vertical lines adjoining this column. To obtain these characters explicitly in a column, either precede them by "\&" or follow them by a space before the usual tab or newline.

Short Horizontal Lines

An input table entry containing only the string "_" is taken to be a single line as wide as the contents of the column. It is not extended to meet adjoining lines.

Vertically Spanned Items

An input table entry containing only the character string "\^" indicates that the table entry immediately above spans downward over this row. It is equivalent to the table format keyletter.

Text Blocks

In order to include a block of text as a table entry, precede it by T{ and follow it by T}. Thus the sequence

```
. . . T{
block of
text
T} . . .
```

is the way to enter, as a single entry in the table, something that cannot conveniently be typed as a simple string between tabs. Note that the T} end delimiter must begin a line; additional columns of data may follow after a tab on the same line. If more than twenty text blocks are used in a table, various limits in the **troff** program are likely to be exceeded, producing diagnostics such as "too many string/macro names" or "too many number registers."

Text blocks are pulled out from the table, processed separately by **troff**, and replaced in the table as a solid block. If no line length is specified in the block of text itself, or in the table format, the default is to use $L \text{ times } C / (N+1)$ where L is the current line length, C is the number of table columns spanned by the text, and N is the total number of columns in the table. The other parameters used in setting the block of text are those in effect at the beginning of the table. These include the effect of the .TS macro and any table format specifications of size, spacing and font, using the p, v and f modifiers to the column keyletters. Commands within the text block itself are also recognized. However, **troff** commands within the table data but not within the text block do not affect that block.

Note the following limitations. Although any number of lines may be present in a table, only the first 200 lines are used in calculating the widths of the various columns. A multipage table may be arranged as several single-page tables if this proves to be a problem. Other difficulties with formatting may arise because in the calculation of column widths all table entries are assumed to be in the font and size being used when the .TS command was encountered. Not included in the calculation are font and size changes indicated in the table format section and within the table data. Therefore, although arbitrary **troff** requests may be sprinkled in a table, care must be taken to avoid confusing the width calculations; use requests such as .ps with care.

Additional Command Lines

If the format of a table must be changed after many similar lines, as with sub-headings or summarizations, the .T& (table continue) command can be used to change column parameters. The outline of such a table input is:

```
.TS
options ;
format .
data
...
.T&
format .
data
.T&
format .
data
.TE
```

Using this procedure, each table line can be close to its corresponding format line. It is not possible to change the number of columns, the space between columns, the global options such as box, or the selection of columns to be made equal width.

Invoking Tbl

You can run **tbl** on a simple table with the command

> tbl *input-file* | troff

but for more complicated use, where there are several input files, and they contain equations and **mm** commands as well as tables, the normal command would be

> tbl *file-1 file-2* ... | eqn | troff -mm

The usual options may be used on the **troff** and **eqn** commands. The usage for **nroff** is similar to that for **troff.**

For the convenience of users employing line printers without adequate driving tables or post-filters, there is a special -TX command line option to **tbl** which produces output that does not have fractional line motions in it. The only other command line option recognized by **tbl** is -mm which fetches the **mm** macro packages.

When you are using both **eqn** and **tbl** on the same file, **tbl** should be used first. 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 (e.g. when you are using the **eqn** delim command), **tbl** must be first or the output will be scrambled. (See Chapter 8, "Formatting Mathematics.") You must also be cautious of using equations in n-style columns; this is nearly always wrong, since **tbl** attempts to split numerical format items into two parts and this is not possible with equations. Give the delim(xx) **tbl** option instead; this prevents splitting of numerical columns within the delimiters. For example, if the **eqn** delimiters are $$, giving delim($$) a numerical column such as "1245 $+- 16$" will be divided after 1245, not after 16.

Tbl limits tables to twenty columns; however, use of more than 16 numerical columns may fail because of limits in **troff,** producing the "too many number registers" message. **Troff** number registers used by **tbl** must be avoided by the user within tables; these include two-digit names from 31 to 99, and names of the forms #x, x+, x|, ^x, and x-, where x is any lowercase letter. The names ##, #-, and #^ are also used in certain circumstances. To conserve number register names, the n and a formats share a register; hence the restriction that they may not be used in the same column.

For aid in writing macros, **tbl** defines a number register TW which 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. To assist in laying out multipage boxed tables the macro T# is defined to produce the bottom lines and side lines of a boxed table, and then invoked at its end. By using this macro in the page footer a multipage table can be boxed. In particular, the

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

    .TS H

a line of the form

    .TH

must be given in the table after any table heading (or at the start if none). Material up to the .TH is placed at the top of each page of table; the remaining lines in the table are placed on several pages as required.

Examples

Here are some examples illustrating features of **tbl**. The symbol ⓣ in the input represents a tab character.

Input:

    .TS
    box;
    c c c
    l l l.
    Language ⓣAuthors ⓣRuns on

    Fortran ⓣMany ⓣAlmost anything
    PL/1 ⓣIBM ⓣ360/370
    C ⓣBTL ⓣ11/45,H6000,370
    BLISS ⓣCarnegie-Mellon ⓣPDP-10,11
    IDS ⓣHoneywell ⓣH6000
    Pascal ⓣStanford ⓣ370
    .TE

Output:

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

Input:

```
.TS
allbox;
c s s
c c c
n n n.
AT&T Common Stock
Year (T)Price (T)Dividend
1971 (T)41-54 (T)$260
2 (T)41-54 (T)270
3 (T)46-55 (T)287
4 (T)40-53 (T)324
5 (T)45-52 (T)340
6 (T)51-59 (T)95*
.TE
* (first quarter only)
```

Output:

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

* (first quarter only)

Input:

```
.TS
box;
c s s
c | c | c
l | l | n.
Major New York Bridges
=
Bridge (T)Designer (T)Length

Brooklyn (T)J A Roebling (T)1595
Manhattan (T)G Lindenthal (T)1470
Williamsburg (T)L L Buck (T)1600

Queensborough (T)Palmer & (T)1182
(T)Hornbostel

(T)(T)1380
Triborough (T)O H Ammann (T)_
(T)(T)383

Bronx Whitestone (T)O H Ammann (T)2300
Throgs Neck (T)O H Ammann (T)1800

George Washington (T)O H Ammann (T)3500
.TE
```

Output:

Major New York Bridges		
Bridge	Designer	Length
Brooklyn	J.A. Roebling	1595
Manhattan	G. Lindenthal	1470
Williamsburg	L.L. Buck	1600
Queensborough	Palmer & Hornbostel	1182
Triborough	O.H. Ammann	1380
		383
Bronx Whitestone	O.H. Ammann	2300
Throgs Neck	O.H. Ammann	1800
George Washington	O.H. Ammann	3500

Input:

```
.TS
c c
np-2 | n | .
(T)Stack
(T)_
1 (T)46
(T)_
2 (T)23
(T)_
3 (T)15
(T)_
4 (T)65
(T)_
5 (T)21
(T)_
.TE
```

Output:

	Stack
1	46
2	23
3	15
4	6.5
5	2.1

Input:

```
.TS
box;
L L L
L L _
L L | LB
L L _
L L L.
january (T)february (T)march
april (T)may
june (T)july (T)Months
august (T)september
october (T)november (T)december
.TE
```

Output:

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

Input:

```
.TS
box;
cfB s s s.
Composition of Foods

.T&
c | c s s
c | c s s
c | c | c | c.
Food (T)Percent by Weight
\^ (T)
\^ (T)Protein (T)Fat (T)Carbo-
\^ (T)\^ (T)\^ (T)hydrate

.T&
l | n | n | n.
Apples (T).4 (T).5 (T)13.0
Halibut (T)18.4 (T)5.2 (T). . .
Lima beans (T)7.5 (T).8 (T)22.0
Milk (T)3.3 (T)4.0 (T)5.0
Mushrooms (T)3.5 (T).4 (T)6.0
Rye bread (T)9.0 (T).6 (T)52.7
.TE
```

Output:

Composition of Foods			
Food	Percent by Weight		
	Protein	Fat	Carbo-hydrate
Apples	.4	.5	13.0
Halibut	18.4	5.2	...
Lima beans	7.5	.8	22.0
Milk	3.3	4.0	5.0
Mushrooms	3.5	.4	6.0
Rye bread	9.0	.6	52.7

Input:

```
.TS
allbox;
cfI s  s
c   cw(1i)  cw(1i)
lp9 lp9 lp9.
New York Area Rocks
Era (T)Formation (T)Age (years)
Precambrian (T)Reading Prong (T)>1 billion
Paleozoic (T)Manhattan Prong (T)400 million
Mesozoic(T)T{
.na
Newark Basin, incl.
Stockton, Lockatong, and Brunswick
formations; also Watchungs
and Palisades.
T} (T)200 million
Cenozoic (T)Coastal Plain (T)T{
On Long Island 30,000 years;
Cretaceous sediments redeposited
by recent glaciation.
.ad
T}
.TE
```

Output:

New York Area Rocks		
Era	Formation	Age (years)
Precambrian	Reading Prong	>1 billion
Paleozoic	Manhattan Prong	400 million
Mesozoic	Newark Basin, incl. Stockton, Lockatong, and Brunswick formations; also Watchungs and Palisades.	200 million
Cenozoic	Coastal Plain	On Long Island 30,000 years; Cretaceous sediments redeposited by recent glaciation.

Summary of tbl Commands

Command	Meaning
a A	Alphabetic subcolumn
allbox	Draws box around all items
b B	Boldface item
box	Draws box around table
c C	Centered column
center	Centers table in page
doublebox	Doubled box around table
e E	Equal width columns
expand	Makes table full line width
f F	Font change
i I	Italic item
l L	Left adjusted column
n N	Numerical column
nnn	Column separation
p P	Point size change
r R	Right adjusted column
s S	Spanned item
t T	Vertical spanning at top
tab (*x*)	Change data separator character
$fat roman "T{" ~~ fat roman "T}"$	Text block
v V	Vertical spacing change
w W	Minimum width value
.*xx*	Included *troff* command
|	Vertical line
||	Double vertical line
^	Vertical span
\^	Vertical span
=	Double horizontal line
$fat "_"$	Horizontal line
$fat "\_"$	Short horizontal line

Introduction

In the previous chapter you were introduced to the **tbl** program, a special preprocessing formatting program which helps you design and create professional-looking tables in documents. This chapter describes another preprocessor: **eqn**, a program that simplifies the task of formatting complex mathematical equations and printing special symbols. Unless you need to use mathematical equations or special symbols in your documents, you can postpone or skip reading about **eqn**.

Like **tbl**, **eqn** is a "preprocessor"--that is, you must embed commands in the text as you are preparing it, along with **mm** macros and **nroff/troff** commands. The **eqn** macros are then translated by the **eqn** program into **nroff/troff** commands, without altering either the body of the text or other formatting commands. The file is processed through the **nroff** or **troff** programs themselves to produce final output.

The uses of **eqn** are fairly specialized. **Eqn** offers you precise control over line spacing, which is suitable to formulas and subscripting, necessary for documents in such fields as chemistry and physics. Special character sets, such as the Greek alphabet, are also available.

Mathematical equations are notoriously difficult to format by conventional typesetting methods. With the help of the XENIX program **eqn**, however, you will quickly learn to use **troff** to typeset mathematical equations directly to a phototypesetter. **Eqn** employs a language which is quite easy to use, even if you know little about either mathematics or typesetting. The formatting commands resemble ordinary English words (e.g., over, lineup, bold, union), and the format is specified much as you might describe an equation in conversation. In a half hour or so, you should be able to learn enough of the language to set

$$\lim_{x \to \pi/2} (\tan x) = 1$$

or

$$G(z) = e = \exp\left(\sum_{k \geq 1} \frac{S\ z}{k}\right) = \prod_{k \geq 1} e^{S_k z/k}$$

$$= \left(1 + S_1 z + \frac{S_1 z}{2!} + \cdots\right)\left(1 + \frac{S z^2}{2} + \frac{S_2 z}{2 \cdot 2!} + \cdots\right) \cdots$$

$$= \sum_{m \geq 0} \left(\sum_{\substack{k,\ k\ \cdots\ ,k \geq 0 \\ k + 2k + \cdots + mk = m}} \frac{S_1^k}{1\ k\ !} \frac{S_2^k}{2\ k\ !} \cdots \frac{S_m^k}{m\ k\ !}\right) z^m$$

The same mathematics formatting commands may be used with the XENIX formatter **nroff** to format mathematical expressions for lineprinters. For compatibility with **nroff**, invoke the program **neqn** instead of **eqn.** The same lineprinter limitations (inability to change font and point size, and do variable spacing, etc.) apply to any text output with **neqn.** The resulting output from **neqn**, however, is usually adequate for proofreading.

As you work with **eqn,** remember that the **eqn** program itself knows relatively little about mathematics. In particular, mathematical symbols like +, -, ×, and parentheses have no special meanings. **Eqn** will set anything that looks like an equation, regardless of whether it makes sense mathematically.

To use **eqn** on your XENIX system, type

    eqn *file* | troff -mm

This command line processes *file* with **eqn,** then pipes the resulting output file to the **troff** program.

Displayed Equations

To tell **eqn** where a mathematical expression begins and ends, surround it with the commands .EQ and .EN. Thus, if you type the lines

    .EQ
    x=y+z
    .EN

your output will look like:

$x=y+z$

The .EQ and .EN are not processed by **eqn.** If you want to specify centering, numbering, or other formatting features for your mathematical text, you will need to enter the appropriate formatting commands in your text. If you want, you can add **nroff/troff** commands, but it is far simpler to use **mm.** **Mm** provides commands which allow you to center, indent, left-justify and number equations.

You can give the .EQ command an argument that is treated as an arbitrary equation number which will be placed in the right margin. For example, the input

    .EQ 7
    x = f(y/2) + y/2
    .EN

produces the output

$x = f(y/2) + y/2$ 7

Basic Mathematical Constructions

This section describes how **eqn** can be used to handle the following frequently used mathematical constructions:

- Subscripts and superscripts

- Grouping

- Fractions

- Square roots

- Summation and integrals

Subscripts and Superscripts

To get subscripts and superscripts into mathematical text, use *sub* and *sup*. For example, the following

> x sup 2 + y sub k

produces

> $x^2 + y_k$

Eqn supplies all the commands for size changes and vertical motions to make the output look right. The words sub and sup must be surrounded by spaces. For example:

> x sub2

will give you *xsub*2 instead of x_2. Furthermore, don't forget to leave a space or a tilde to mark the end of a subscript or superscript. Note that if you use an expression like

> y = (x sup 2)+1

you will get

> $y = (x^2) + 1$

instead of

> $y = (x^2) + 1$

Subscripted subscripts and superscripted superscripts can also be created. The following

> x sub i sub 1

produces

> x_i

A subscript and superscript on the same object are printed one above the other if the subscript comes first. For example,

> x sub i sup 2

produces

> x_i^2

Other than in this special case, sub and sup group to the right, so x sup y sub z means x^z, not x.

Braces for Grouping

Normally, the end of a subscript or superscript is marked simply by a blank, tab, or tilde. If you need to produce a subscript or superscript with blanks in it, you can use braces ({ }) to mark the beginning and end of the subscript or superscript. For example:

> e sup {i omega t}

produces:

> $e^{i\omega t}$

Braces can always be used to force **eqn** to treat an expression as a unit, or just to make your intention perfectly clear. When you use braces:

> x sub {i sub 1} sup 2

produces

> x_{i_1}

The same text without braces:

    x sub i sub 1 sup 2

produces

    $x_i{}^2$

Braces can occur within braces if necessary:

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

results in

    $e^{i \pi +}$

The general rule is that anywhere you could use a single item like x, you could substitute any complicated expression, if you enclose it in braces. Positioning and size will be taken care of by **eqn.**

You will need to make sure you have the right number of braces. If for some reason you need to print braces, enclose them in double quotations ("), e.g., "{".

Fractions

To make a fraction, use the word "over". For example:

    a+b over 2c =1

produces

    $$\frac{a+b}{2c} = 1$$

The line is made the right length and positioned automatically. You can use braces to make clear what goes over what:

    {alpha + beta} over {sin (x)}

is

    $$\frac{a+\beta}{\sin(x)}$$

If you have both an "over" and a "sup" in the same expression, **eqn** does the "sup" before the "over", so

    -b sup 2 over pi

is $\frac{-b^2}{\pi}$ instead of $-b^{\frac{2}{\pi}}$.

The rules of precedence that control which operation will be done first are summarized at the end of this chapter. If you are in doubt, however, use braces to make clear what you mean.

Square Roots

To draw a square root, use "sqrt". For example

    sqrt a+b + 1 over sqrt {ax sup 2 +bx+c}

produces

$$\sqrt{a+b}+\ \frac{1}{\sqrt{ax^2+bx+c}}$$

You should note, however, that the square roots of tall quantities often do not look good. A square root big enough to cover the quantity is too dark and heavy. For example

    sqrt {a sup 2 over b sub 2}

produces

$$\sqrt{\frac{a^2}{b_2}}$$

You are better off writing big square roots as the power 1/2. For example, you could use

    (a sup 2 /b sub 2) sup half

to produce

$$(a^2/b)^{\frac{1}{2}}$$

Summation and Integrals

Summations, integrals, and similar constructions can be produced with **eqn**. For example

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

produces

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

Braces are used here to indicate where the upper part $i=\infty$ begins and ends. No braces were necessary for the lower part $i=0$, because it contained no blanks. Braces never hurt, and if the from and to parts contain any blanks, you must use braces around them. The from and to parts are optional, but if both are used, they have to occur in that order.

Other useful characters can replace the sum, including:

    int   prod   union   inter

These become, respectively,

    $\int$ Π $\cup$ $\cap$

The expression before the "from" can be anything, including an expression in braces. The from-to expression can often be used in unexpected ways. For example

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

produces

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

Complex Mathematical Constructions

This section describes how to use **eqn** to produce more complicated mathematical constructions, including piles and matrices, often surrounded by brackets, parentheses or bars.

Big Brackets, Parentheses, and Bars

To get big brackets ([ ]), braces ({ }), parentheses (( )), and bars (||) around things, use the left and right commands. For example

    left { a over b + 1 right }
    ~=~ left (c over d right)
    + left [e right]

produces

$$\left\{ \frac{a}{b} + 1 \right\} = \left(\frac{c}{d} \right) + \left[e \right]$$

The resulting brackets are big enough to cover whatever they enclose. Other characters can be used besides these, but they probably won't look very good.

One exception is the floor and ceiling characters. For example

>   left floor x over y right floor
>   < = left ceiling a over b right ceiling

produces

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

Please note that braces are typically bigger than brackets and parentheses, because the number of pieces is incremented by two (three, five, seven, etc.) while the number of pieces in a bracket is incremented by one (two, three, etc.). Also, big left and right parentheses often look poor, because of character set limitations.

The right part may be omitted: a left expression need not have a corresponding right expression. If the right part is omitted, put braces around the thing you want the left bracket to encompass. Otherwise the resulting brackets may be too large. If you want to omit the left part, things are more complicated, because technically you can't have a right without a corresponding left. Instead you have to say

>   left " " ..... right)

The left " " means a "left nothing". This satisfies the rules without affecting your output.

Piles

There is a facility for making vertical piles of things with several variants. For example:

>   A ~=~ left [
>   pile { a above b above c }
>   ~~ pile { x above y above z }
>   right]

will produce

$$A = \begin{bmatrix} a & x \\ b & y \\ c & z \end{bmatrix}$$

You can have as many elements in a pile as you want. They will be centered one above another, at the right height for most purposes. The keyword above is used to separate the pieces; braces are used around the entire list. The elements of a pile can be as complicated as needed, and may even contain more piles.

Three other forms of pile exist: "lpile" makes a pile with the elements left-justified; "rpile" makes a right-justified pile; and "cpile" makes a centered pile, just like pile. The vertical spacing between the pieces is somewhat larger for 1-, r- and cpiles than it is for ordinary piles. For example

>   roman sign (x)~=~
>   left {
>   lpile {1 above 0 above -1}
>   ~~ lpile
>   {if~x>0 above if~x=0 above if~x<0}

creates the pile

$$\text{sign } (x) = \begin{cases} 1 \text{ if } x>0 \\ 0 \text{ if } x=0 \\ -1 \text{ if } x<0 \end{cases}$$

Note that the left brace has no matching right one.

Matrices

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

$$x_i \quad x^2$$

$$y_i \quad y^2$$

use

>   matrix {
>   ccol { x sub i above y sub i }
>   ccol { x sup 2 above y sup 2 }
>   }

This produces a matrix with two centered columns. The elements of the columns are then listed just as for a pile, each element separated by the word above. You can also use lcol or rcol to left or right adjust columns. Each column can be separately adjusted, and there can be as many columns as you like.

The reason for using a matrix instead of two adjacent piles is that if the elements of the piles do not all have the same height, they will not line up properly. A matrix forces them to line up, because it looks at the entire structure before deciding what spacing to use. A word of warning about matrices: each column must have the same number of elements in it.

Lining Up Equations

Sometimes it is necessary to line up a series of equations at some horizontal position, such as at an equal sign. This is done with two operations called "mark" and "lineup." The word mark may appear once at any place in an equation. It remembers the horizontal position where it appeared. Successive equations can contain one occurrence of the word lineup. The place where lineup appears is made to line up with the place marked by the previous mark if at all possible. Thus, for example, you can say

```
.EQ I
x+y mark = z
.EN
.EQ I
x lineup = 1
.EN
```

to produce

$$x+y = z$$
$$x = 1$$

I
I

Note that mark does not look ahead, so

```
x mark =1
...
x+y lineup =z
```

will not work, because there is not room for the x+y part after the mark remembers where the x is.

Layout and Design of Mathematical Text

The following sections describe the format and layout control features of **eqn.**

Input Spaces

Eqn ignores spaces and newlines within an expression. If you have any of the following equations between .EQ and .EN commands,

```
x=y+z
```

or

```
x = y + z
```

or

```
x = y
  + z
```

they will all produce the same output:

$x=y+z$

Therefore, use spaces and newlines freely to make your input equations readable and easy to edit.

Output Spaces

To get extra spaces into the output, use a tilde (~) for each space you want:

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

This produces

$x = y + z$

You can also use a caret (^), which produces a space half the width of a tilde. Tabs may be used to position pieces of an expression, but the tab stops must be set with the **troff** tab (.ta) command.

Spaces Between Special Sequences

If you need to separate a special sequence of characters, you will have to make this clear to **eqn**. You can either surround a special sequence with ordinary spaces, tabs, or newlines, or make special words stand out by surrounding them with tildes or carets, as in the following:

```
x~=~2~pi~int~sin~(~omega~t~)~dt
```

The tildes not only separate the words sin, omega, etc., but also add extra spaces, one space per tilde:

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

Special words can also be separated by braces ({ }) and double quotation marks (").

Symbols, Special Names, and Greek Characters

Eqn knows some mathematical symbols, some mathematical names, and the Greek alphabet. For example,

>   x=2 pi int sin (omega t)dt

produces

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

Here you need input spaces to tell **eqn** that int, pi, sin and omega are separate entities that should get special treatment. The sin, digit 2, and parentheses are set in Roman type instead of italic; pi and omega are translated into Greek; int becomes the integral sign.

When in doubt, leave spaces around separate parts of the input. A common error is to type f(pi) without leaving spaces on both sides of the pi. If you do this, **eqn** does not recognize pi as a special word, and it appears as *f(pi)* instead of $f(\pi)$. A complete list of **eqn** names appears at the end of this chapter. You can also use **troff** names for anything **eqn** doesn't know about.

Size and Font Changes

By default, equations are set in 10-point type; standard mathematical conventions determine which characters are in Roman and which are in italic. If you are dissatisfied with the default sizes and fonts, you can change them using the commands size *n* and roman, italic, bold and fat. Like sub and sup, size and font changes affect only what follows immediately and then revert to the default. Thus

>   bold x y

is

>   **x**y

and

>   size 14 bold x = y +
>   size 14 {alpha + beta}

gives

> $\mathbf{X}=y+\alpha+\beta$

You can use braces if you want to apply a change to something more complicated than a single letter. For example, you can change the size of an entire equation with

    size 12 { ... }

Legal sizes are: 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; for example, you can say

    size~+2

to make the size two points bigger, or

    size~-3

to make it three points smaller. The advantage of this method is that you do not need to know what the current size is.

If you are using fonts other than Roman, italic and bold, you can say font X where X is a one character **troff** name or number for the font. However, since **eqn** is designed for Roman, italic and bold, other fonts may not give quite as good an appearance.

The fat operation takes the current font and widens it by overstriking: fat grad is ∇ and fat {x sub i} is x_i.

If an entire document is to be in a nonstandard size or font, you need not write out a size and font change for each equation. Instead, you can set a "global" size or font which thereafter affects all equations. At the beginning of any equation, you might say, for instance,

    .EQ
    gsize 16
    gfont R
    ...
    .EN

to set the size to 16 points and the font to Roman. In place of R, you can use any **troff** font name. The size after gsize can be a relative change with + or -.

Generally, gsize and gfont will appear at the beginning of a document but they can also appear throughout a document: the global font and size can be changed as often as needed. 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. Don't forget to reset the global size at the end of the footnote.

Diacritical Marks

There are several words that produce diacritical marks on top of letters:

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

The diacritical mark is automatically placed at the correct height. The "bar" and "under"are made the right length for the entire construct, as in $\overline{x+y+z}$; other marks are centered.

Quoted Text

Any input entirely within quotes ("...") is not subject to any of the font changes and spacing adjustments normally done by the equation setter. This provides a way to do your own spacing and adjusting if needed. For example

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

produces

   $sin(x)$ + sin (x)

Quotation marks are also used to get braces and other **eqn** keywords printed. For example

   "{ size alpha }"

produces

   { size alpha }

Similarly

   roman "{ size alpha }"

produces

   { size alpha }

The construction " " can be used as a place-holder when **eqn** syntax requires something, but you don't actually want anything in your output. For example, to make

    H e

you can't just type

    sup 2 roman He

because a sup has to be a superscript on something. Thus you must say

    " " sup 2 roman He

To get a literal quotation mark, use the sequence \".

Local Motions

Although **eqn** tries to get most things at the right place on the paper, it isn't perfect, and occasionally you will need to tune the output to make it just right. Small extra horizontal spaces can be obtained with tildes (~) and carets (^). You can also say "back *n*" and "fwd *n*" to move small distances horizontally. The *n* is the distance to be moved in 1/100 em units (an em is about the width of the letter m). Thus "back 50" moves back about half the width of an m. Similarly you can move things up or down with "up *n*" and "down *n*." As with sub or sup, the local motions affect the next entry in the input. This can be a complex expression, as long as it is enclosed in braces.

In-line Equations

In a mathematical document it is often necessary to follow mathematical conventions in the body of the text, as well as in display equations. For example, you may need to make variable names like x italic. Although this could be done by surrounding the appropriate parts with .EQ and .EN, the continual repetition of .EQ and .EN is a nuisance. Furthermore, this implies a displayed equation.

Eqn provides a shorthand for short in-line expressions. You can define two characters to mark the left and right ends of an in-line equation, and then type expressions right in the middle of text lines. To set both the left and right characters to percent signs, for example, add to the beginning of your document the three lines

    .EQ
    delim %%
    .EN

Having done this, you create text like

> Let %alpha sub i% be the primary variable, and let %beta% be zero. Then we can show that %x sub 1% is %>=0%.

> Let α be the primary variable, and let β be zero. Then we can show that x is ≥ 0.

This works as you might expect: spaces, newlines, and so on are significant in the text, but not in the equation part itself. Multiple equations can occur in a single input line.

Enough room is left before and after a line that contains in-line expressions that something like $\sum\limits_{i=1}^{n} x_i$ does not interfere with the lines surrounding it.

To turn off the delimiters, use:

```
.EQ
delim off
.EN
```

Do not use braces, tildes, carets, or double quotation marks as delimiters; these have special meanings.

Definitions

Eqn allows you to give a frequently used string of characters a name, and thereafter just type the 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 retyping it each time by defining it like this:

> define xy 'x sub i sub 1 + y sub i sub 1'

This makes xy a shorthand for whatever characters occur between the single quotation marks in the definition. You can use any character instead of quotation marks to indicate the ends of the definition, so long as that character does not appear inside the definition.

You can use xy like this:

```
.EQ
f(x) = xy ...
.EN
```

Each occurrence of *xy* will expand into the string of characters you defined. Be careful to leave spaces or their equivalent around the name when you actually use it, so **eqn** will be able to identify it as special.

There are several things to watch out for. First, although definitions can use previous definitions, as in:

```
.EQ
define  xi  ' x sub i '
define  xi1  ' xi sub 1 '
.EN
```

don't define something in terms of itself. You cannot use

```
define  X  ' roman X '
```

because this defines X in terms of itself. If you say

```
define  X  ' roman "X" '
```

however, the quotation marks protect the second X, and everything works fine.

Eqn keywords can be also be redefined. You can make / mean over by saying

```
define  /  ' over '
```

or redefine over as / with

```
define  over ' / '
```

If you need to print a symbol one way on a terminal and another way on the typesetter, it is sometimes worth defining a symbol differently for **neqn** and **eqn**. This can be done with "ndefine" and "tdefine." A definition made with ndefine only takes effect if you are running **neqn**. If you use tdefine, the definition only applies for **eqn**. Names defined with "define" apply to both **eqn** and **neqn**.

Invoking eqn

To print a document that contains mathematics on the typesetter, use

```
eqn files | troff
```

If there are any **troff** options, place them after the **troff** part of the command. For example,

```
eqn files | troff -mm files
```

To print equations on a lineprinter or similar device, use

```
neqn files | nroff -mm files
```

The language for equations recognized by **neqn** is identical to that of **eqn,** although of course the output is more restricted.

Eqn and **neqn** can be used with the **tbl** program for setting tables that contain mathematics. Use **tbl** before **eqn** like this:

```
tbl files  |  eqn  |  troff -mm
tbl files  |  neqn  |  nroff -mm
```

Sample Equation

Now that you are familiar with the features of **eqn,** here is the complete input and output for the three display equations at the beginning of this chapter:

```
.EQ I
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
.EQ I
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
.EQ I
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 1} } 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
```

Output:

$$G(z) = e = \exp\left(\sum_{k \geq 1} \frac{S_k z}{k}\right) = \prod_{k \geq 1} e^{S_k z/k}$$

$$= \left(1 + S_1 z + \frac{S_1 z}{2!} + \cdots\right)\left(1 + \frac{S\ z}{2} + \frac{S_2 z}{2 \cdot 2!} + \cdots\right) \cdots$$

$$= \sum_{m \geq 0} \left(\sum_{\substack{k,\ k\ \cdots\ ,k \geq 0 \\ k + 2k + \cdots + mk = m}} \frac{S_1^k}{1\ k\ !} \frac{S_2^k}{2\ k\ !} \cdots \frac{S_m^k}{m\ k\ !}\right) z^m$$

Error Messages

If you make a mistake in an equation, such as leaving out a brace or having one too many braces or having a sup with nothing before it, **eqn** will respond with the message

syntax error between lines x and y, *file*

where x and y are the lines between which the trouble occurred, and *file* is the name of the file in question. The line numbers are only approximate, so check nearby lines as well. You will receive self-explanatory messages if you leave out a quotation mark or try to run **eqn** on a nonexistent file.

If you want to check a document before actually printing it try:

eqn *files* >/dev/null

This will throw away the output but print the error messages.

If you use something like dollar signs as delimiters, it is easy to leave one out. The program **eqncheck** checks for misplaced or missing dollar signs and similar errors.

In-line equations are limited in size because of an internal buffer in **troff**. If you get the message "word overflow", you have exceeded this limit. If you print the equation as a display this message will usually go away. The message "line" overflow indicates you have exceeded an even bigger buffer. The only cure for this is to break the equation into two separate ones.

Also, **eqn** does not break equations by itself; you must split long equations up across multiple lines by yourself, marking each by a separate .EQ ....EN sequence. **Eqn** warns about equations that are too long to fit on one line.

Summary of Keywords and Precedences

If you don't use braces around expressions, **eqn** will do operations in the order shown in this list.

dyad	vec	under	bar	tilde	hat	dot	dotdot
fwd	back	down	up				
fat	roman	italic	bold	size			
sub	sup	sqrt	over				
from	to						

These operations group to the left:

over sqrt left right

All others group to the right.

Digits, parentheses, brackets, punctuation marks, and these mathematical words are converted to Roman font when encountered:

sin	cos	tan	sinh	cosh	tanh	arc
max	min	lim	log	ln	exp	
Re	Im	and	if	for	det	

>=	$\geq$
<=	$\leq$
==	$\equiv$
!=	$\neq$
+-	$\pm$
->	$\rightarrow$
<-	$\leftarrow$
<<	$\ll$
>>	$\gg$
inf	∞
partial	∂
half	
prime	$'$
approx	$\approx$
nothing	
cdot	$\cdot$
times	$\times$
del	∇
grad	∇
...	$\cdots$
,...,	$, \cdots ,$
sum	Σ
int	$\int$
prod	Π
union	$\cup$
inter	$\cap$

DELTA	Δ	iota	ι
GAMMA	Γ	kappa	κ
LAMBDA	Λ	lambda	λ
OMEGA	Ω	mu	μ
PHI	Φ	nu	ν
PI	Π	omega	ω
PSI	Ψ	omicron	o
SIGMA	Σ	phi	φ
THETA	Θ	pi	π
UPSILON	Υ	psi	ψ
XI	Ξ	rho	ρ
alpha	α	sigma	σ
beta	β	tau	τ
chi	χ	theta	θ
delta	δ	upsilon	υ
epsilon	ε	xi	ξ
eta	η	zeta	ζ
gamma	γ		

These are all the words known to **eqn** except for characters with names:

above	dotdot	italic	rcol	to
back	down	lcol	right	under
bar	dyad	left	roman	up
bold	fat	lineup	rpile	vec
ccol	font	lpile size	~, ^	
col	from	mark	sqrt	{ }
cpile	fwd	matrix	sub	\"...\"
define	gfont	ndefine	sup	
delim	gsize	over	tdefine	
dot	hat	pile	tilde	

This section describes use of the individual commands available in the Text Processing portion of the XENIX 286 Extended System. Reference sections describing the Extended System's programming commands and utilities can be found in the *XENIX 286 C Library Guide* and the *XENIX 286 Programmer's Reference Manual*.

Syntax

Unless otherwise noted, commands described in this reference section accept options and other arguments according to the following syntax:

        name [option...] [cmdarg...]

This syntax is detailed below:

name   The file name or path name of an executable file.

option   A single letter representing a command option. By convention, most options are preceded with a dash. Option letters can sometimes be grouped together as in **-abcd** or specified individually as in **-a -b -c -d.** The method of specifying options depends on the syntax of the individual command. Options specified individually can be given arguments. For example, the **-f** option for many commands often takes a following file name argument.

cmdarg   A path name or other command argument **not** beginning with a dash. It may also be a dash alone by itself, indicating the standard input.

See Also

getopt in "Commands" in the *XENIX 286 Reference Manual*
getopt in "System Functions" in the *XENIX 286 C Library Guide*

Diagnostics

Upon termination, each command returns 2 bytes of status, one supplied by the system and giving the cause for termination, and (in the case of "normal" termination) one supplied by the program. (See **wait** and **exit** in "System Functions" in the *XENIX 286 C Library Guide*.) The former byte is zero for normal termination; the latter is customarily zero for successful execution and nonzero to indicate troubles such as erroneous parameters, bad or inaccessible data, or other inability to cope with the task at hand. It is variously called "exit code," "exit status," and "return code" and is described only where special conventions are involved.

Notes

Many commands do not adhere to the given syntax.

Summary of Commands

Command	Definition	Page
col	Filters reverse linefeeds	A-3
cut	Cuts out selected fields of each line	A-5
cw, cwcheck	Prepares constant-width text for **troff**	A-7
deroff	Removes **nroff/troff, tbl,** and **eqn** constructs	A-11
diction	Checks language usage	A-12
diffmk	Marks differences between files	A-13
eqn, neqn, eqncheck	Formats mathematical text for **troff**	A-15
explain	Corrects language usage	A-19
hyphen	Finds hyphenated words	A-20
mm	Prints documents formatted with the **MM** macros	A-21
mmcheck	Checks the usage of **MM** macros	A-24
mmt	Typesets documents	A-25
neqn	Formats mathematics	A-27
nroff	A text formatter	A-28
paste	Merges lines of files	A-31
prep	Prepares text for statistical processing	A-33
ptx	Generates a permuted index	A-34
soelim	Eliminates **.so**'s from **nroff** input	A-36
spell, spellin, spellout	Finds spelling errors	A-38
style	Analyzes writing characteristics	A-40
tbl	Formats tables for **nroff** or **troff**	A-42
troff	Typesets text	A-45

col - Filters reverse linefeeds.

Syntax

col [-bfxp]

Description

col prepares output from processes such as the text formatter **nroff** for output on devices that limit ot do not allow reverse or half-line motions. The **col** command is typically used to process **nroff** output text that contains tables generated by the **tbl** program. A typical command line might be:

tbl *file* | nroff | col | lpr

col takes the following options:

-b   Assumes the output device in use is not capable of backspacing. In this case, if two or more characters are to appear in the same place, **col** outputs the last character read.

-f   Allows forward half-linefeeds. If this option is not given, **col** accepts half-line motions in its input, but text that would appear between lines is moved down to the next full line. Reverse full and half linefeeds are never allowed with this option.

-x   Prevents conversion of white space to tabs on output. **col** normally converts white space to tabs wherever possible to shorten printing time.

-p   Causes **col** to ignore escape sequences found in its input and pass them to the output as regular characters. Because these characters are subject to overprinting from reverse line motions, the use of this option is discouraged unless the user is fully aware of the position of the escape sequences.

col assumes that the ASCII control characters **SO** (octal 016) and **SI** (octal 017) start and end text in an alternate character set. If you have a reverse linefeed (ESC 7), a reverse half-linefeed (ESC 8), or a forward half-linefeed (ESC 9) within an SI-SO sequence, the ESC 7, 8, and 9 are still recognized as line motions.

On input, the only control characters **col** accepts are space, backspace, tab, return, newline, reverse linefeed (ESC 7), reverse half-linefeed (ESC 8), forward half-linefeed (ESC 9), alternate character start (SI), alternate character end (SO), and vertical tag (VT). The **VT** character is an alternate form of full reverse linefeed, included for compatibility with some earlier programs of this type. All other nonprinting characters are ignored.

See Also

nroff, tbl

Notes

col cannot back up more than 128 lines.

col allows at most 800 characters, including backspaces, on a line.

Local vertical motions that would result in backing up over the first line of the document are ignored. Therefore, the first line must not contain any superscripts.

cut - Cuts out selected fields of each line of a file.

Syntax

cut -c*list* [*file1* *file2* ...]
cut -f*list* [-d*char*] [-s] [*file1* *file2* ...]

Description

Use **cut** to cut out columns from a table or fields from each line of a file. The fields as specified by **list** can have a fixed length (-**c** option), or their length can vary from line to line and be marked with a field delimiter character like **tab** (-**f** option). **cut** can be used as a filter; if no files are given, the standard input is used. The output is standard output.

The meanings of the options are:

list   A comma-separated list of integer field numbers (in increasing order) with optional dash (-) to indicate ranges, as in the -**o** option of **nroff/troff** for page ranges; e.g., **1,4,7; 1-3,8; -5,10** (short for **1-5,10**); or **3**-(short for third through last field).

-**c***list*   The *list* following -**c** (no space) specifies character positions. (For example, -**c1-72** would pass the first 72 characters of each line.)

-**f***list*   The *list* following -**f** is a list of fields assumed to be separated in the file by a delimiter character (see the next entry); e.g., -**f1,7** copies the first and seventh field only. Lines with no field delimiters will be passed through intact (useful for table subheadings), unless -**s** is specified.

-**d***char*   The character following -**d** is the field delimiter (-**f** option only). The default is **tab** if -**d** is not specified. Space or other characters with special meaning to the shell must be quoted.

-**s**   Suppresses lines with no delimiter characters in case of -**f** option. Unless specified, lines with no delimiters will be passed through untouched.

Either the -**c** or -**f** option must be specified.

Hints

Use **grep** to make horizontal "cuts" (by context) through a file, or **paste** to put files together horizontally. To reorder columns in a table, use **cut** and **paste**.

Examples   (

cut -d: -f1,5 /etc/passwd   Maps user IDs to names

who am i | cut -f1 -d" "   Gives you the current login name

See Also

paste

grep in "Commands" in the *XENIX 286 Reference Manual*

Diagnostics

line too long   A line can have no more than 511 characters or fields.

bad list for c/f option   Missing **–c** or **–f** option or incorrectly specified list.   No
                          error occurs if a line has fewer fields than the list calls for.

no fields   The list is empty.

no delimiter   Missing (char) for **–d** option; char not specified.

cw, cwcheck - Prepares constant-width text for **troff.**

Syntax

cw [-lxx] [-rxx] [-fn] [-t] [+t] [-d] [*file* ...]

cwcheck [-lxx] [-rxx] *file* ...

Description

cw prepares **troff** input files that contain text to be typeset in the constant-width (CW) font.

Because the CW font contains a nonstandard set of characters and requires character and interword spacing different from standard fonts, documents that use the CW font must be preprocessed by **cw.** Typical usage is:

> cw *file* | troff ...

The **cwcheck** program checks to see that the left and right delimiters, as well as the **.CW/.CN** pairs, are properly balanced. It prints out all incorrect lines. **cwcheck** is invoked as follows:

> cwcheck [options] *file*

The options for **cw** and **cwcheck** are:

-l*xx*   Designates the one- or two-character string *xx* as the left delimiter; if *xx* is omitted, the left delimiter is undefined, which is the default setting. This option can be used with both **cw** and **cwcheck.**

-r*xx*   Designates the one- or two-character string *xx* as the right delimiter. The left and right delimiters may be different. This option can be used with both **cw** and **cwcheck.**

-f*n*   Mounts the CW font in position *n*; acceptable values for *n* are 1, 2, and 3. This option is useful only at the beginning of a document and can be used only with **cw.**

-t   Turns transparent mode off. This option can be used only with **cw.**

+t   Turns transparent mode on. (This is the default.) This option can be used only with **cw.**

-d   Prints current option settings on standard error in the form of **troff** comment lines. This option is meant for debugging and can be used only with **cw.**

The left and right delimiters perform the same function as the .CW/.CN requests; they are meant, however, to enclose CW words or phrases in running text. cw treats text enclosed by delimiters in the same mannner as text bracketed by .CW/.CN pairs except that while spaces in text bracketed by .CW/.CN pairs have the same width as any other CW character, spaces between delimiters are half as wide, so they have the same width as spaces in the prevailing text. This width is not adjustable.

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

cw recognizes five requests. The requests look like **troff** macros (see the "Examples" section that follows), and they are copied by cw onto its output. Thus, the user can define them as **troff** macros.

The five requests are:

.CW   Marks the start of text to be set in the CW font. .CW takes the same options--in the same format--that are available on the cw command line.

.CN   Marks the end of text to be set in the CW font. .CN takes the same options that are available on the cw command line.

.CD *option(s)*
              Changes delimiters and/or settings of other options; takes the same options that are available on the cw command line.

.CP *arg1 arg2 arg3 ...*
              Sets the odd-numbered arguments in the CW font and the even-numbered arguments in the prevailing font. The arguments are delimited like **troff** macro arguments.

.PC *arg1 arg2 arg3 ...*
              Same as for .CP except that it sets the even-numbered arguments in the CW font and the odd-numbered arguments in the prevailing font. The arguments are delimited like **troff** macro arguments.

Except for the .CD request and the nine special four-character names listed in the table that follows, every character between the .CW and .CN requests is taken literally and output as is. The -t option turns off this feature (called transparent mode) and applies normal **troff** rules to the CW text.

Text set with the CW font resembles the output of terminals and lineprinters. The font is often used to typeset examples of programs and computer output in documents such as user manuals and programming texts. The CW font contains the 94 printing ASCII characters:

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

It also contains 8 nonASCII characters represented by 4-character **troff** names:

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

The hyphen is a synonym for the minus sign (-).

Examples

The following are typical definitions of the **.CW** and **.CN** macros. They are to be used with the **MM** macro package.

```
.de CW                  Begins definition
.DS I                   Display start, indented
.ps 9                   9-point type
.vs 10.5p               Vertical spacing 10.5 points
.ta 16m/3u 32m/3u 48m/3u 64m/3u 80m/3u 96m/3u
                        Sets tabs
..                      Ends definition

.de CN                  Begins definition
.ta 0.5i 1i 1.5i 2i 2.5i 3i 3.5i 4i 4.5i 5i 5.5i 6i
                        Resets tabs
.vs                     Resets vertical spacing
.ps                     Resets point size
.DE                     Ends display
..                      Ends definition
```

When set in running text, the CW font is, by default, set in the same point size as the rest of the text. In displayed matter, it can often be profitably set one point smaller than the prevailing point size. When the CW font is set in 9-point type, there are 12 characters per inch.

If a document that contains CW text also contains tables and equations, the order of preprocessing should be **cw, tbl,** and **eqn.** Usually, the tables contained in such documents will not contain any CW text, although it is possible to have elements of the table set in the CW font. Care must be taken that **tbl** format information is not modified by **cw.** Attempts to set equations in the CW font are not likely to be either pleasing or successful.

Files

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

See Also

eqn, mmt, tbl, troff

Warning

Text preprocessed by **cw** must be set on a typesetter equipped with the CW font.

Notes

Don't use periods (.) or backslashes (\) as delimiters.

Certain CW characters don't fit gracefully with certain Times Roman characters, such as a CW ampersand (&) followed by a Times Roman comma (,). In such cases, use **troff** half- and quarter-spaces. For more information, see the "Notes" section of the **troff** reference entry later in this appendix.

deroff - Removes **nroff/troff**, **tbl**, and **eqn** constructs.

Syntax

deroff [-w] [-mx] [*files*]

Description

deroff reads each of the *files* in sequence; removes all **troff** requests, macros, backslash constructs, **eqn** constructs (between .**EQ** and .**EN** lines, and between delimiters), and **tbl** descriptions; and writes the remainder of the file on the standard output. **deroff** follows chains of included files (.**so** and .**nx troff** commands). If a file has already been included, a .**so** naming that file is ignored, and a .**nx** naming that file terminates execution. If no input file is given, **deroff** reads the standard input.

The -**m** option may be followed by an **m, s,** or **l.** The resulting -**mm** or -**ms** option causes the **MM** or **MS** macros to be interpreted so that only running text (i.e., no text from macro lines) is output. The -**ml** option forces the -**mm** option and also causes deletion of lists associated with the **mm** macros. This option is used by the **diction** command.

The -**w** option outputs is a word list, one "word" per line, with all other characters deleted. Otherwise, the output follows the original, with the deletions just mentioned. In text, a "word" is any string that contains at least two letters and is composed of letters, digits, ampersands (&), and apostrophes ('). In a macro call, however, a word is a string that begins with at least two letters and contains a total of at least three letters. Delimiters are any characters other than letters, digits, apostrophes, and ampersands. Trailing apostrophes and ampersands are removed from words.

See Also

diction, eqn, style, tbl, troff

Notes

deroff is not a complete **troff** interpreter, so it can be confused by subtle constructs. Most such errors result in too much rather than too little output.

The -**ml** option does not handle nested lists correctly.

deroff also removes words of two or fewer letters in lines that begin with macro calls or **troff** requests.

diction - Checks language usage.

Syntax

diction [-ml] [-mm] [-n] [-f *patternfile*] *file* ...

Description

diction finds all sentences in a document that contain phrases from a data base of bad or wordy diction. On output, each phrase is enclosed within brackets. Because **diction** runs **deroff** before looking at the text, formatting header files should be included as part of the input. The options are:

-ms   Overrides the default macro package **MM.**

-ml   Causes **deroff** to skip lists. Should be used if the document contains many lists of nonsentences.

-f*patternfile*
            A user-supplied *patternfile* of words and phrases is used in addition to the default file.

Credit

This utility was developed at the University of California at Berkeley and is used with permission.

See Also

deroff, explain

Notes

Use of nonstandard formatting macros may cause incorrect sentence breaks.

The -n option cannot be specified by itself.

diffmk – Marks differences between files.

Syntax

diffmk *name1 name2 name3*

Description

diffmk compares two versions of a file and creates a third file that includes "change mark" commands for **nroff** or **troff**. *name1* and *name2* are the old and new versions of the file, respectively. **diffmk** generates *name3*, which contains the lines of *name2* plus inserted formatter "change mark" (**.mc**) requests. When *name3* is formatted, changed, or inserted, the altered text is indicated by a vertical bar (|) at the right margin of each line. The position of deleted text is shown by a single asterisk (*).

The **diffmk** command will produce listings of C (or other) programs with changes marked. A typical command line for such use is:

    diffmk old.c new.c tmp; nroff macs tmp | pr

where the file **macs** contains:

    .pl 1
    .ll 77
    .nf
    .eo
    .nc

The **.ll** request might specify a different line length depending on the nature of the program being printed. The **.eo** and **.nc** requests are probably needed only for C programs.

If the characters | and * are inappropriate, a copy of **diffmk** can be edited to change them. (**diffmk** is a shell procedure.)

See Also

nroff

diff in "Commands" in the *XENIX 286 Reference Manual*

Notes

Aesthetic considerations may dictate manual adjustment of some output. File differences involving only formatting requests may produce undesirable output; that is, replacing **.sp** by **.sp 2** will produce a change mark on the preceding or following line of output.

eqn, neqn, eqncheck - Formats mathematical text for **nroff** or **troff.**

Syntax

eqn [-d*xy*] [-p*n*] [-s*n*] [-f*font*] [*file* ...]

neqn [-d*xy*] [-p*n*] [-s*n*] [-f*font*] [*file* ...]

eqncheck [files]

Description

eqn is a **troff** preprocessor for typesetting mathematical text on a phototypesetter; **neqn** is used with **nroff** for setting mathematical text on typewriter-like terminals. Usage is normally one of the following or its equivalent:

> eqn *files* | troff
> neqn *files* | nroff

If no files are specified, these programs read from the standard input.

The options are:

-d*xy*   Reduces subscripts and superscripts *n* points from the previous size; the default reduction is 3 points.

-s*n*   Sets **eqn** delimiters to characters *x* and *y*.

-p*n*   Changes the point size within **eqn** delimiters to *n*.

-f*font*   Changes the font within **eqn** delimiters to *font*.

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

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

Tokens within **eqn** are separated by spaces, tabs, newlines, braces, double quotation marks, tildes, and carets. Braces ({}) are used for grouping; generally speaking, anywhere a single character such as an *x* could appear, a complicated construction enclosed in braces might be used instead. Tilde (~) represents a full space in the output, caret (^) half as much.

Subscripts and superscripts are produced with the keywords **sub** and **sup**. Thus **x sub j** makes

$$x_j$$

a sub k sup 2 produces

$$a_k^2$$

while

$$e^{x^2+y^2}$$

is made with **e sup {x sup 2 + y sup 2}**. Fractions are made by using the keyword **over**: **a over b** yields

$$\frac{a}{b}$$

The keyword **sqrt** makes square roots: **1 over sqrt {ax sup 2+bx+c}** results in

$$\frac{1}{\sqrt{ax^2+bx+c}}$$

The keywords **from** and **to** introduce lower and upper limits, as in this example.

$$\lim_{n\to\infty} \sum_0^n x_i$$

is made with **lim from {n -> inf } sum from 0 to n x sub i**. Left and right brackets, braces, etc., of the right height are made with **left** and **right**. **left [x sup 2 + y sup 2 over alpha right] ~=~ 1** produces

$$\left[x^2 + \frac{y^2}{a} \right] = 1$$

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

Vertical "piles" are made with **pile, lpile, cpile,** and **rpile**:   **pile {a above b above c}** produces

$$a$$
$$b$$
$$c$$

Piles may have arbitrary numbers of elements; **lpile** left-justifies, **pile** and **cpile** center (but with different vertical spacing), and **rpile** right justifies. Matrices are made with **matrix**: **matrix { lcol { x sub i above y sub 2 } ccol { 1 above 2 } }** produces

$$x_i \quad 1$$
$$y_2 \quad 2$$

In addition, there is **rcol** for a right-justified column.

Diacritical marks are made with **dot, dotdot, hat, tilde, bar, vec, dyad,** and **under:** **x dot = f(t) bar** produces

$$\dot{x} = \overline{f(t)}$$

y dotdot bar ~=~ n under is

$$\overline{\ddot{y}} = \underline{n}$$

and **x vec ~=~ y dyad** is

$$\vec{x} = \overleftrightarrow{y}$$

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

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

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

Shorthands may be defined or existing keywords redefined with **define.** For example

    define thing % replacement`%

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

Keywords such as **sum** (Σ), **int** ($\int$), **inf** (∞), and shorthands such as >= ($\geq$), != ($\neq$), and ->
($\rightarrow$) are recognized by **eqn**. Greek letters are spelled out in the desired case, as in **alpha**
(α) or **GAMMA** (Γ). Mathematical words such as **sin, cos,** and **log** are made Roman
automatically. **troff** four-character escapes such as **\(dd** ($\ddagger$) and **\(bs** ($\oslash$) may be used
anywhere. Strings enclosed in double quotation marks are passed through untouched;
this permits keywords to be entered as text, and it can be used to communicate with
troff when all else fails.

See Also

mm, mmt, tbl, troff

Notes

To embolden digits, parentheses, etc., it is necessary to surround them with double
quotation marks. See the "Notes" section of the **troff** entry later in this appendix.

explain - Corrects language usage.

Syntax

explain

Description

explain interactively reports on language usage. It suggests alternatives to phrases found with the **diction** command.

Credit

This utility was developed at the University of California at Berkeley and is used with permission.

See Also

deroff, diction

hyphen - Finds hyphenated words.

Syntax

hyphen *file* ...

Description

hyphen finds all the hyphenated words in files and prints them on the standard output. If no arguments are given, the standard input is used. Thus **hyphen** may be used as a filter.

Notes

hyphen doesn't properly deal with hyphenated **italic** (i.e., underlined) words; it will often miss them completely.

hyphen occasionally gets confused, but with no ill effects other than extra output.

mm - Prints documents formatted with the **MM** macros.

Syntax

mm [*options*] [*files*]

Description

mm can be used to print documents with **nroff** and the **MM** text formatting macro package. It has options to specify preprocessing by **tbl** and/or **neqn** and postprocessing by various terminal-oriented output filters. The proper pipelines and the required arguments and flags for **nroff** and **MM** are generated depending on the options selected.

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

-c Causes **mm** to invoke **col.**

-e Causes **mm** to invoke **neqn.**

-t Causes **mm** to invoke **tbl.**

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

-y Causes **mm** to use the noncompacted version of the macros.

-T *name*
  Specifies the output device. The default device is "lp," the lineprinter.

Other supported devices include:

-T300   DASI (DTC, GSI) 300

-T300s   DASI 300s

-T450   DASI 450 (same as Diablo 1620)

-T300-12   DASI 300 at 12-pitch

-T300s-12   DASI 300s at 12-pitch

-T450-12   DASI 450 at 12-pitch

-T33	TTY 33; invokes **col** automatically
-Tdumb	Terminal types with no special features; invokes **col** automatically
-T37	TTY 37
-T735	TI 735; invokes **col** automatically
-T745	TI 735; invokes **col** automatically
-T43	TTY 43; invokes **col** automatically
-T40/2	Teletype model 40/2; invokes **col** automatically
-T40/4	Teletype model 40/4; invokes **col** automatically
-T2631	HP 2631 series lineprinter; invokes **col** automatically
-T2631e	HP 2631 series lineprinter, expanded mode; invokes **col** automatically
-T2631c	HP 2631 series lineprinter, compressed mode; invokes **col** automatically
-T42	ADM 42; invokes **col** automatically
-T31	TTY 31; invokes **col** automatically
-T1620	Diablo 1620 (same as DASI 450)
-T1620-12	Diablo 1620 at 12 pitch

The two command lines below are equivalent:

```
mm - t - rC3 - 12 ghh*
tbl file* | nroff - cm - h - rC3
```

mm reads the standard input when a dash is specified instead of any file names. (Mentioning other files together with the dash can lead to disaster.) This option allows **mm** to be used as a filter. For example:

```
cat dws | mm -
```

Hints

1. **mm** invokes **nroff** with the **–h** flag. With this flag, **nroff** assumes that the terminal has tabs set every 8 character positions.

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

3. If you use the **–s** option of **nroff** (to stop between pages of output), use linefeed rather than return or newline to restart the output. The **–s** option of **nroff** does not work if the **–c** option of **mm** is used or if **mm** automatically invokes **col.** (See the **–c** option described earlier.)

Use the **mmcheck** program to check the contents of **mm** source files for errors in usage of the macros.

See Also

col, env, eqn, mmt, mmcheck, nroff, tbl

profile in "Files" in the *XENIX 286 Reference Manual*

Diagnostics

mm: no input file   None of the arguments is a readable file, and **mm** has not been used as a filter.

mmcheck - Checks usage of **MM** macros.

Syntax

[files]

Description

mmcheck checks files for usage of the **MM** formatting macros and some **eqn** constructions. Appropriate messages are produced. The program skips all directories, and if no file name is given, the standard input is read.

See Also

col, env, eqn, mm, mmt, nroff, tbl

profile in "Files" in the *XENIX 286 Reference Manual*

Diagnostics

mmcheck-unreadable files cause the message "Cannot open filename." The remaining output of the program is diagnostic of the source file.

mmt - Typesets documents.

Syntax

mmt [*options*] [*file*]

Description

mmt uses the MM macro package. It has options to specify preprocessing by **tbl** and **eqn.** The proper pipelines and the required arguments and flags for **troff** and for the macro packages are generated depending on the options selected.

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

-e   Causes these commands to invoke **eqn.**

-t   Causes these commands to invoke **tbl.**

-a   Invokes the -a option of **troff.**

-y   Causes **mmt** to use the noncompacted version of the macros. (See the entry for **mm** earlier in this appendix.)

When a dash (-) is specified, **mmt** reads the standard input rather than file names.

Hints

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

See Also

eqn, mm, tbl, troff

env, environ in "Commands," **profile** in "Files" in the *XENIX 286 Reference Manual*

Diagnostics

mmt: no input file   None of the arguments is a readable file, and the command is not used as a filter.

neqn - Formats mathematics.

Syntax

neqn [-d*xy*] [-f*n*] [*file*] ...
checkeq [*file*] ...

Description

neqn is an **nroff** preprocessor for formatting mathematics on terminals and printers; **eqn** is its counterpart for typesetting with **troff**. Its form is usually:

neqn *file* ... | nroff

If no files are specified, these programs read from the standard input. A line beginning with **.EQ** marks the start of an equation; the end of an equation is marked by a line beginning with **.EN**. These lines are not altered, so they may be defined in macro packages to get centering, numbering, etc. It is also possible to set two characters as delimiters; subsequent text between delimiters is also treated as **neqn** input. Delimiters may be set to characters x and y with the command line argument -d*xy* or (more commonly) with **delim** *xy* between **.EQ** and **.EN**. The left and right delimiters may be identical. Delimiters are turned off by **delim off**. All text that is neither between delimiters nor between **.EQ** and **.EN** is passed through untouched. Fonts can be changed globally in a document with **gfont** *n*, or with the command line argument -f*n*.

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

Tokens within **neqn** are separated by spaces, tabs, newlines, braces, double quotation marks, tildes, or carets. Braces are used for grouping; generally speaking, anywhere a single character like an x could appear, a complicated construction enclosed in braces may be used instead. Tilde (~) represents a full space in the output, caret (^) half as much.

See Also

eqn, checkeq, troff, tbl

Notes

To embolden digits, parentheses, etc., it is necessary to surround them by double quotation marks, as in **bold "12.3"**.

nroff - A text formatter.

Syntax

nroff [*option* ...] [*file* ...]

Description

nroff formats text in the named *file*. Part of the **nroff/troff** family of text formatters, **nroff** is used to format files for output to a lineprinter or daisy wheel printer, whereas **troff** formats files for a phototypesetter.

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

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

-n*N*   Numbers the first generated page, *N*.

-s*N*   Stops every *N* pages. **nroff** will halt prior to every *N* pages (default *N*=1) to allow paper loading or changing and will resume upon receipt of a newline.

-m*name*   Prepends the macro file **/usr/lib/tmac/tmac.***name* to the input files.

-ra*N*   Sets register *a* (one-character) to *N*.

-i   Reads standard input after the input files are exhausted.

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

-e   Produces equally spaced words in adjusted lines, using full terminal resolution.

-h   Uses output tabs during horizontal spacing to speed output and reduce output character count. Tab settings are assumed to be every 8 nominal character widths.

-T*device*   Specifies the output device. The default device is **lp**, the lineprinter.

Other supported devices include:

-T300	DASI (DTC, GSI) 300
-T300s	DASI 300s
-T450	DASI 450 (same as Diablo 1620)
-T300-12	DASI 300 at 12-pitch
-T300s-12	DASI 300s at 12-pitch
-T450-12	DASI 450 at 12-pitch
-T33	TTY 33; invokes **col** automatically
-Tdumb	Terminal types with no special features; invokes **col** automatically
-T37	TTY 37
-T735	TI 735; invokes **col** automatically
-T745	TI 745; invokes **col** automatically
-T43	TTY 43; invokes **col** automatically
-T40/2	Teletype model 40/2; invokes **col** automatically
-T40/4	Teletype model 40/4; invokes **col** automatically
-T2631	HP 2631 series lineprinter; invokes **col** automatically
-T2631e	HP 2631 series lineprinter, expanded mode; invokes **col** automatically
-T2631c	HP 2631 series lineprinter, compressed mode; invokes **col** automatically
-T42	ADM 42; invokes **col** automatically
-T31	TTY 31; invokes **col** automatically
-T35	TTY 35; invokes **col** automatically
-T1620	Diablo 1620 (same as DASI 450)
-T1620-12	Diablo 1620 at 12-pitch

Files

/usr/lib/suftab	Suffix hyphenation tables
/tmp/ta*	Temporary file
/usr/lib/tmac/tmac.*	Standard macro files
/usr/lib/term/*	Terminal driving tables

See Also

col, eqn, tbl, troff

paste - Merges lines of files.

Syntax

paste *file1 file2 ...*

paste -d*list file1 file2 ...*

paste -s [-d*list*] *file1 file2 ...*

Description

In the first two forms, **paste** concatenates corresponding lines of the given input files *file1, file2,* etc. It treats each file as a column or columns of a table, and it pastes them together horizontally (parallel merging). It is the counterpart of **cat,** which concatenates vertically, that is, one file after the other. In the last form shown in the "Syntax" section, **paste** subsumes the function of an older command with the same name by combining subsequent lines of the input file (serial merging). In all cases, lines are "glued" together with the tab character or with characters from an optionally specified list. Output is to the standard output, so it can be used as the start of a pipe or as a filter, provided - is used in place of a file name.

The meanings of the options are:

-d   Without this option, the newline characters of each but the last file (or last line, in the case of the -s option) are replaced by a tab character. This option allows replacement of the tab character by one or more alternate characters.

list   One or more characters immediately following -d replace the default tab as the line concatenation character. The list is used circularly; that is, when exhausted, it is reused. In parallel merging (no -s option), the lines from the last file are always terminated with a newline character. The list may contain these special escape sequences: \n (newline), \t (tab), \\ (backslash), and \0 (empty string, not a null character). Quoting may be necessary, if characters have special meaning to the shell. (For example, to get one backslash, use -d"\\\\" .)

-s   This option merges subsequent lines rather than lines from each input file. Use a tab for concatenation unless a list is specified with the -d option. Regardless of the **list,** the very last character of the file must be a newline.

-       May be used in place of any file name to read a line from the standard input. (There is no prompting.)

Examples

ls | paste -d" " -   Lists directory in one column

ls | paste - - - -   Lists directory in four columns

paste -s -d"\t\n" file   Combines pairs of lines into lines

See Also

cut

grep and **pr** in "Commands" in the *XENIX 286 Reference Manual*

Diagnostics

line too long   Output lines are restricted to 511 characters.

too many files   Except for the –s option, no more than 12 input files may be specified.

prep – Prepares text for statistical processing.

Syntax

prep [-dio] *file* ...

Description

prep reads each *file* in sequence and writes it on the standard output, one "word" to a line. A word is a string of alphabetic characters and embedded apostrophes, delimited by space or punctuation. Hyphenated words are broken apart; hyphens at the ends of lines are removed and the hyphenated parts are joined. Strings of digits are discarded.

The following option letters may appear in any order:

-d   Prints the word number (in the input stream) with each word.

-i   Takes the next file as an "ignore" file. These words will not appear in the output. (They will be counted for purposes of the -d count.)

-o   Takes the next **file** as an "only" file. Only these words will appear in the output. (All other words will also be counted for the -d count.)

-p   Includes punctuation marks (single nonalphanumeric characters) as separate output lines. The punctuation marks are not counted for the -d count.

The *ignore* and *only* files contain words, one per line.

See Also

deroff

ptx - Generates a permuted index.

Syntax

ptx [*options*] [*input* [*output*]]

Description

ptx generates a permuted index to file *input* on file *output* (standard input and output default). It has three phases. The first performs the permutation, generating one line for each keyword in an input line. The keyword is rotated to the front. In the second phase, the permuted file is sorted. In the third phase, the sorted lines are rotated so that the keyword comes at the middle of each line. **ptx** produces output in the form:

        .xx "tail" "before keyword" "keyword and after" "head"

where **.xx** is assumed to be an **nroff** or **troff** macro provided by the user. The **"before keyword"** and **"keyword and after"** fields incorporate as much of the line as will fit around the keyword when it is printed. **"tail"** and **"head"**, at least one of which is always the empty string, are wrapped-around pieces small enough to fit in the unused space at the opposite end of the line.

The following options can be applied:

- f	Folds uppercase and lowercase letters for sorting.
- t	Prepares the output for the phototypesetter.
-w *n*	Uses the next argument, *n*, as the length of the output line. The default line length is 72 characters for **nroff** and 100 for **troff**.
- g *n*	Uses the next argument, *n*, as the number of characters that **ptx** will reserve in its calculations for each gap among the 4 parts of the line as finally printed. The default gap is 3 characters.
- o *only*	Uses as keywords only the words given in the *only* file.
- i *ignore*	Does not use as keywords any words given in the *ignore* file. If the –i and –o options are missing, use **/usr/lib/eign** as the *ignore* file.
- b *break*	Uses the characters in the *break* file to separate words. Tab, newline, and space characters are always used as break characters.
-r	Takes the leading nonblank character of each input line to be a reference identifier (that is, referring to a page or chapter), separate from the text of the line. Attaches that identifier as a fifth field on each output line.

Files

/bin/sort

/usr/lib/eign

Notes

Line length counts do not account for overstriking or proportional spacing.

Lines that contain tildes (~) are not handled correctly because **ptx** uses that character internally.

soelim – Eliminates **.so**'s from **nroff** input.

Syntax

soelim [*file* ...]

Description

soelim reads the specified files or the standard input and performs the textual inclusion implied by the **nroff** directives of the form

    .so somefile

when they appear at the beginning of input lines. This is useful since programs such as **tbl** do not normally do this. **soelim** allows the placement of individual tables in separate files to be run as a part of a large document.

Note that inclusion can be suppressed by using a single quotation mark (') instead of a dot (.). For example

    'so /usr/lib/tmac.s

Example

A typical usage of **soelim** might be

    soelim exum?.n | tbl | nroff -mm | col | lpr

See Also

nroff, troff

Credit

This utility was developed at the University of California at Berkeley and is used with permission.

Notes

Exactly one blank must precede the file name; no blanks may follow it.   Lines of the form

.if t .so /usr/lib/macros.t

mean that **.so** statements embedded in the text are expanded.

spell, spellin, spellout - Finds spelling errors.

Syntax

spell [*options*] [*files*]

/usr/lib/spell/spellin [*list*]

/usr/lib/spell/spellout [-d] *list*

Description

spell collects words from named files and looks them up in a spelling list. Words that neither occur among nor are derivable from words in the spelling list (by applying certain inflections, prefixes, or suffixes) are printed on the standard output. If no files are named, words are collected from the standard input.

spell ignores most **troff, tbl,** and **eqn** constructions.

Under the **-v** option, all words not literally in the spelling list are printed, and plausible derivations from the words in the spelling list are indicated.

Under the **-b** option, British spelling is checked. Besides preferring "centre," "colour," "speciality," "travelled," etc., this option insists upon the -ise suffix in words like "standardise."

Under the **-x** option, every plausible stem is printed with an equals sign (=) for each word.

The spelling list is based on many sources, and while more haphazard than an ordinary dictionary, it is also more effective with respect to proper names and popular technical words. Coverage of the specialized vocabularies of biology, medicine, and chemistry is light.

Pertinent auxiliary files may be specified by name arguments, indicated in the next section, "Files," with their default settings. The stop list filters out misspellings (for example, **thier=thy-y+ier**) that would otherwise pass.

Two routines help maintain the hash lists used by **spell**. Both expect a list of words, one per line, from the standard input. **spellin** adds the words on the standard input to the pre-existing list and places a new list on the standard output. If no list is specified, the new list is created from scratch. **spellout** looks up each word read from the standard input and prints on the standard output those that are missing from (or, with the **-d** option, present in) the hash list.

Files

D_SPELL=/usr/lib/spell/hlist[ab]   Hashed spelling lists, American and British

S_SPELL=/usr/lib/spell/hstop   Hashed stop list

/tmp/spell.   Temporary

/usr/lib/spell/spellprog   Program

See Also

deroff, eqn, tbl, troff

sed, sort, and **tee** in "Commands" in the *XENIX 286 Reference Manual*

Notes

The spelling list's coverage is uneven. You may wish to monitor the output for several months and make your own additions. Typically, local additions are kept in a separate local dictionary that is added to the hashed list via **spellin.**

By default, logging of errors to **/usr/lib/spell/spellhlist** is turned off.

D_SPELL and **S_SPELL** can be overridden by placing alternate path definitions in your environment.

style - Analyzes characteristics of a document.

Syntax

style [-ml] [-mm] [-a] [-e] [-l num] [-r num]
     [-p] [-P] *file ...*

Description

style analyzes the characteristics of the writing style of a document. It reports on readability, sentence length and structure, word length and usage, verb type, and sentence openers. Because **style** runs **deroff** before looking at the text, formatting header files should be included as part of the input. The default macro package -**ms** may be overridden with the flag -**mm**. The flag -**ml**, which causes **deroff** to skip lists, should be used if the document contains many lists of nonsentences. The other options are used to locate sentences with certain characteristics.

-**a**   Prints all sentences with their length and readability index

-**e**   Prints all sentences that begin with an expletive

-**p**   Prints all sentences that contain a passive verb

-**l***num*
       Prints all sentences longer than *num*

-**r***num*
       Prints all sentences whose readability index is greater than *num*

-**P**   Prints parts of speech of the words in the document

Credit

This utility was developed at the University of California at Berkeley and is used with permission.

See Also

deroff, diction

Notes

Use of nonstandard formatting macros may cause incorrect sentence breaks.

tbl - Formats tables for **nroff** or **troff.**

Syntax

tbl [-TX] [*files*]

Description

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

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

center	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.
doublebox	Encloses the table in a double box.
allbox	Encloses each item of the table in a box.
tab (*x*)	Uses the character *x* instead of a tab to separate items in a line of input data.

The global options, if any, are terminated with a semicolon (;).

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

c	Centers item within the column.
r	Right-adjusts item within the column.
l	Left-adjusts item within the column.
n	Numerically adjusts item in the column. Unit positions of numbers are aligned vertically.
s	Spans previous item on the left into this column.
a	Centers longest line in this column and then left-adjusts all other lines in this column with respect to that centered line.
^	Spans down previous entry in this column.
_	Replaces this entry with a horizontal line.
=	Replaces this entry with a double horizontal line.

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

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

If a data line consists of only an underscore (_) or if an equals sign (=) occurs, then a single or double line, respectively, is drawn across the table at that point. If a single item in a data line consists of only an underscore or equals sign, then that item is replaced by a single or double line.

The –TX option forces **tbl** to use only full vertical line motions, making the output more suitable for devices such as lineprinters, which cannot generate partial vertical line motions.

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

Example

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

```
.TS
center box ;
cB s s
cl | cl s
^ | c c
l | n n .
Household Population
_
Town→Households
→Number→Size
=
Bedminster→789→3.26
Bernards Twp.→3087→3.74
Bernardsville→2018→3.30
Bound Brook→3425→3.04
Bridgewater→7897→3.81
Far Hills→240→3.19
.TE
```

yields:

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

See Also

eqn, mm, mmt, troff

Notes

See the "Notes" section of the entry for **troff** later in this appendix.

troff - Typesets text.

Syntax

troff [*options*] [*files*]

Description

Part of the **nroff/troff** family of text formatters, **troff** can be used to output text to a number of phototypesetters and laser printers. **nroff** formats text contained in *files* (standard input by default) for printing on typewriter-like devices and lineprinters.

An argument consisting of a lone dash (-) is taken to be a file name corresponding to the standard input.

These options, which may appear in any order, but must appear before the files, are common to both **troff** and **nroff**:

-o*list*   Prints only pages whose page numbers appear in the **list** of numbers and ranges, separated by commas. A range **N-M** means pages **N** through **M**; an initial **-N** means from the beginning to page **N**; and a final **N-** means from **N** to the end. (See the "Notes" section later in this entry.)

-n*N*   Numbers first generated page *N*.

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

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

-i   Reads standard input after *files* are exhausted.

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

-z   Prints only messages generated by **.tm** (terminal message) requests.

-m*name*   Prepends to the input *files* the noncompacted (ASCII text) macro file **/usr/lib/tmac/tmac.***name*.

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

-k*name*   Compacts the macros used in this invocation of **nroff/troff,** placing the output in files **[dt].**name in the current directory.

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

-h   Uses output tabs during horizontal spacing to speed output and reduce output character count. Tab settings are assumed to be every 8 nominal character widths.

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

These options are for use with **troff** only:

-t   Directs output to the standard output instead of the phototypesetter.

-f   Refrains from feeding out paper and stopping phototypesetter at the end of the run.

-w   Waits until phototypesetter is available, if it is currently busy.

-b   Reports whether the phototypesetter is busy or available. No text processing is done.

-a   Sends a printable ASCII approximation of the results to the standard output.

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

-T*name*   Uses font width tables for device *name* (the font tables are found in **/usr/lib/font/name/*).** Currently, no *names* are supported.

Files

/usr/lib/suftab	Suffix hyphenation tables
/tmp/ta#	Temporary file
/usr/lib/tmac/tmac	Standard macro files and pointers
/usr/lib/macros/*	Standard macro files
/usr/lib/term/*	Terminal driving tables for **nroff**
/usr/lib/font/*	Font width tables for **troff**

See Also

eqn, tbl

for **nroff** only -- **col, mm**

for **troff** only -- **mmt**

Notes

nroff/troff uses Eastern Standard Time; as a result, depending on the time of the year and on your local time zone, the date that **nroff/troff** generates may be off by one day.

When **nroff/troff** is used with the *-olist* option inside a pipeline (for example, with one or more of **cw, eqn,** and **tbl**) it may cause a harmless "broken pipe" diagnostic if the last page of the document is not specified in **list.**

REQUEST FOR READER'S COMMENTS

Intel's Technical Publications Departments attempt to provide publications that meet the needs of all Intel product users. This form lets you participate directly in the publication process. Your comments will help us correct and improve our publications. Please take a few minutes to respond.

Please restrict your comments to the usability, accuracy, organization, and completeness of this publication. If you have any comments on the product that this publication describes, please contact your Intel representative. If you wish to order publications, contact the Literature Department (see page ii of this manual).

1. Please describe any errors you found in this publication (include page number).

_____

_____

_____

_____

2. Does this publication cover the information you expected or required? Please make suggestions for improvement.

_____

_____

_____

_____

3. Is this the right type of publication for your needs? Is it at the right level? What other types of publications are needed?

_____

_____

_____

_____

4. Did you have any difficulty understanding descriptions or wording? Where?

_____

_____

_____

5. Please rate this publication on a scale of 1 to 5 (5 being the best rating). _____

NAME _____ DATE _____

TITLE _____

COMPANY NAME/DEPARTMENT _____

ADDRESS_____

CITY _____ STATE _____ ZIP CODE _____

(COUNTRY)

Please check here if you require a written reply. ☐

WE'D LIKE YOUR COMMENTS . . .

This document is one of a series describing Intel products. Your comments on the back of this form will help us produce better manuals. Each reply will be carefully reviewed by the responsible person. All comments and suggestions become the property of Intel Corporation.

intel®